Android进程包活全部方法

自己曾经也在这个问题上伤过脑经，前几日刚好有一个北京的哥们在QQ说在做IM类的项目，问我进程保活如何处理比较恰当，决定去总结一下，网上搜索一下进程常驻的方案好多好多，但是很多的方案都是不靠谱的或者不是最好的，结合很多资料，今天总结一下Android进程保活的一些方案，都附有完整的实现源码，有些可能你已经知道，但是有些你可能是第一次听说，（1像素Activity，前台服务，账号同步，Jobscheduler,相互唤醒，系统服务捆绑，如果你都了解了，请忽略）经过多方面的验证，Android系统中在没有白名单的情况下做一个任何情况下都不被杀死的应用是基本不可能的，但是我们可以做到我们的应用基本不被杀死，如果杀死可以马上满血复活，原谅我讲的特别含蓄，毕竟现在的技术防不胜防啊，不死应用还是可能的。

有几个问题需要思考，系统为什么会杀掉进程，杀的为什么是我的进程，这是按照什么标准来选择的，是一次性干掉多个进程，还是一个接着一个杀，保活套路一堆，如何进行进程保活才是比较恰当......如果这些问题你还还存在，或许这篇文章可以解答。

## 一、进程初步了解

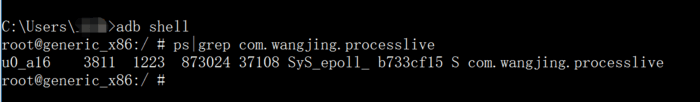
每一个Android应用启动后至少对应一个进程，有的是多个进程，而且主流应用中多个进程的应用比例较大



Paste\_Image.png

1、如何查看进程解基本信息

对于任何一个进程，我们都可以通过adb shell ps|grep <package\_name>的方式来查看它的基本信息



| 值 | 解释 |
| --- | --- |
| u0\_a16 | USER 进程当前用户 |
| 3881 | 进程ID |
| 1223 | 进程的父进程ID |
| 873024 | 进程的虚拟内存大小 |
| 37108 | 实际驻留”在内存中”的内存大小 |
| com.wangjing.processlive | 进程名 |

### 2、进程划分

Android中的进程跟封建社会一样，分了三流九等，Android系统把进程的划为了如下几种（重要性从高到低），网上多位大神都详细总结过（备注：严格来说是划分了6种）。

2.1、前台进程(Foreground process)

场景：

某个进程持有一个正在与用户交互的Activity并且该Activity正处于resume的状态。

某个进程持有一个Service，并且该Service与用户正在交互的Activity绑定。

某个进程持有一个Service，并且该Service调用startForeground()方法使之位于前台运行。

某个进程持有一个Service，并且该Service正在执行它的某个生命周期回调方法，比如onCreate()、 onStart()或onDestroy()。

某个进程持有一个BroadcastReceiver，并且该BroadcastReceiver正在执行其onReceive()方法。

用户正在使用的程序，一般系统是不会杀死前台进程的，除非用户强制停止应用或者系统内存不足等极端情况会杀死。

2.2、可见进程(Visible process)

场景：

拥有不在前台、但仍对用户可见的 Activity（已调用 onPause()）。

拥有绑定到可见（或前台）Activity 的 Service

用户正在使用，看得到，但是摸不着，没有覆盖到整个屏幕,只有屏幕的一部分可见进程不包含任何前台组件，一般系统也是不会杀死可见进程的，除非要在资源吃紧的情况下，要保持某个或多个前台进程存活

2.3、服务进程(Service process)

场景

某个进程中运行着一个Service且该Service是通过startService()启动的，与用户看见的界面没有直接关联。

在内存不足以维持所有前台进程和可见进程同时运行的情况下，服务进程会被杀死

2.4、后台进程(Background process)

场景：

在用户按了"back"或者"home"后,程序本身看不到了,但是其实还在运行的程序，比如Activity调用了onPause方法

系统可能随时终止它们，回收内存

2.5、空进程(Empty process)

场景：

某个进程不包含任何活跃的组件时该进程就会被置为空进程，完全没用,杀了它只有好处没坏处,第一个干它!

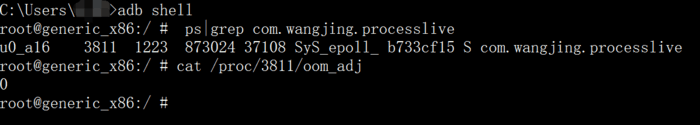
### 3、内存阈值

上面是进程的分类，进程是怎么被杀的呢？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。那这个不足怎么来规定呢，那就是内存阈值，我们可以使用cat /sys/module/lowmemorykiller/parameters/minfree来查看某个手机的内存阈值。

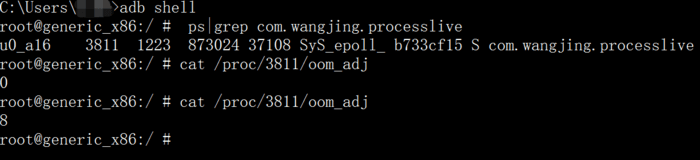
https://upload-images.jianshu.io/upload_images/1836169-756cdcc354f8ff5b.png?imageMogr2/auto-orient/strip%7CimageView2/2/w/700

注意这些数字的单位是page. 1 page = 4 kb.上面的六个数字对应的就是(MB): 72,90,108,126,144,180，这些数字也就是对应的内存阀值,内存阈值在不同的手机上不一样，一旦低于该值,Android便开始按顺序关闭进程. 因此Android开始结束优先级最低的空进程，即当可用内存小于180MB(46080\*4/1024)。

读到这里，你或许有一个疑问，假设现在内存不足，空进程都被杀光了，现在要杀后台进程，但是手机中后台进程很多，难道要一次性全部都清理掉？当然不是的，进程是有它的优先级的，这个优先级通过进程的adj值来反映，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，adj值定义在com.android.server.am.ProcessList类中，这个类路径是${android-sdk-path}\sources\android-23\com\android\server\am\ProcessList.java。oom\_adj的值越小，进程的优先级越高，普通进程oom\_adj值是大于等于0的，而系统进程oom\_adj的值是小于0的，我们可以通过cat /proc/进程id/oom\_adj可以看到当前进程的adj值。



看到adj值是0，0就代表这个进程是属于前台进程，我们按下Back键，将应用至于后台，再次查看



adj值变成了8，8代表这个进程是属于不活跃的进程，你可以尝试其他情况下，oom\_adj值是多少，但是每个手机的厂商可能不一样，oom\_adj值主要有这么几个，可以参考一下。

| adj级别 | 值 | 解释 |
| --- | --- | --- |
| UNKNOWN\_ADJ | 16 | 预留的最低级别，一般对于缓存的进程才有可能设置成这个级别 |
| CACHED\_APP\_MAX\_ADJ | 15 | 缓存进程，空进程，在内存不足的情况下就会优先被kill |
| CACHED\_APP\_MIN\_ADJ | 9 | 缓存进程，也就是空进程 |
| SERVICE\_B\_ADJ | 8 | 不活跃的进程 |
| PREVIOUS\_APP\_ADJ | 7 | 切换进程 |
| HOME\_APP\_ADJ | 6 | 与Home交互的进程 |
| SERVICE\_ADJ | 5 | 有Service的进程 |
| HEAVY\_WEIGHT\_APP\_ADJ | 4 | 高权重进程 |
| BACKUP\_APP\_ADJ | 3 | 正在备份的进程 |
| PERCEPTIBLE\_APP\_ADJ | 2 | 可感知的进程，比如那种播放音乐 |
| VISIBLE\_APP\_ADJ | 1 | 可见进程 |
| FOREGROUND\_APP\_ADJ | 0 | 前台进程 |
| PERSISTENT\_SERVICE\_ADJ | -11 | 重要进程 |
| PERSISTENT\_PROC\_ADJ | -12 | 核心进程 |
| SYSTEM\_ADJ | -16 | 系统进程 |
| NATIVE\_ADJ | -17 | 系统起的Native进程 |

备注：（上表的数字可能在不同系统会有一定的出入）

根据上面的adj值，其实系统在进程回收跟内存回收类似也是有一套严格的策略，可以自己去了解，大概是这个样子的，oom\_adj越大，占用物理内存越多会被最先kill掉，OK，那么现在对于进程如何保活这个问题就转化成，如何降低oom\_adj的值，以及如何使得我们应用占的内存最少。

一、进程保活方案

1、开启一个像素的Activity

据说这个是手Q的进程保活方案，基本思想，系统一般是不会杀死前台进程的。所以要使得进程常驻，我们只需要在锁屏的时候在本进程开启一个Activity，为了欺骗用户，让这个Activity的大小是1像素，并且透明无切换动画，在开屏幕的时候，把这个Activity关闭掉，所以这个就需要监听系统锁屏广播，我试过了，的确好使，如下。

public class MainActivity extends AppCompatActivity {

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

}

}

如果直接启动一个Activity，当我们按下back键返回桌面的时候，oom\_adj的值是8，上面已经提到过，这个进程在资源不够的情况下是容易被回收的。现在造一个一个像素的Activity。

public class LiveActivity extends Activity {

public static final String TAG = LiveActivity.class.getSimpleName();

public static void actionToLiveActivity(Context pContext) {

Intent intent = new Intent(pContext, LiveActivity.class);

intent.setFlags(Intent.FLAG\_ACTIVITY\_NEW\_TASK);

pContext.startActivity(intent);

}

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

Log.d(TAG, "onCreate");

setContentView(R.layout.activity\_live);

Window window = getWindow();

//放在左上角

window.setGravity(Gravity.START | Gravity.TOP);

WindowManager.LayoutParams attributes = window.getAttributes();

//宽高设计为1个像素

attributes.width = 1;

attributes.height = 1;

//起始坐标

attributes.x = 0;

attributes.y = 0;

window.setAttributes(attributes);

ScreenManager.getInstance(this).setActivity(this);

}

@Override

protected void onDestroy() {

super.onDestroy();

Log.d(TAG, "onDestroy");

}

}

为了做的更隐藏，最好设置一下这个Activity的主题，当然也无所谓了

<style name="LiveStyle">

<item name="android:windowIsTranslucent">true</item>

<item name="android:windowBackground">@android:color/transparent</item>

<item name="android:windowAnimationStyle">@null</item>

<item name="android:windowNoTitle">true</item>

</style>

在屏幕关闭的时候把LiveActivity启动起来，在开屏的时候把LiveActivity 关闭掉，所以要监听系统锁屏广播，以接口的形式通知MainActivity启动或者关闭LiveActivity。

public class ScreenBroadcastListener {

private Context mContext;

private ScreenBroadcastReceiver mScreenReceiver;

private ScreenStateListener mListener;

public ScreenBroadcastListener(Context context) {

mContext = context.getApplicationContext();

mScreenReceiver = new ScreenBroadcastReceiver();

}

interface ScreenStateListener {

void onScreenOn();

void onScreenOff();

}

/\*\*

\* screen状态广播接收者

\*/

private class ScreenBroadcastReceiver extends BroadcastReceiver {

private String action = null;

@Override

public void onReceive(Context context, Intent intent) {

action = intent.getAction();

if (Intent.ACTION\_SCREEN\_ON.equals(action)) { // 开屏

mListener.onScreenOn();

} else if (Intent.ACTION\_SCREEN\_OFF.equals(action)) { // 锁屏

mListener.onScreenOff();

}

}

}

public void registerListener(ScreenStateListener listener) {

mListener = listener;

registerListener();

}

private void registerListener() {

IntentFilter filter = new IntentFilter();

filter.addAction(Intent.ACTION\_SCREEN\_ON);

filter.addAction(Intent.ACTION\_SCREEN\_OFF);

mContext.registerReceiver(mScreenReceiver, filter);

}

}

public class ScreenManager {

private Context mContext;

private WeakReference<Activity> mActivityWref;

public static ScreenManager gDefualt;

public static ScreenManager getInstance(Context pContext) {

if (gDefualt == null) {

gDefualt = new ScreenManager(pContext.getApplicationContext());

}

return gDefualt;

}

private ScreenManager(Context pContext) {

this.mContext = pContext;

}

public void setActivity(Activity pActivity) {

mActivityWref = new WeakReference<Activity>(pActivity);

}

public void startActivity() {

LiveActivity.actionToLiveActivity(mContext);

}

public void finishActivity() {

//结束掉LiveActivity

if (mActivityWref != null) {

Activity activity = mActivityWref.get();

if (activity != null) {

activity.finish();

}

}

}

}

现在MainActivity改成如下

public class MainActivity extends AppCompatActivity {

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

final ScreenManager screenManager = ScreenManager.getInstance(MainActivity.this);

ScreenBroadcastListener listener = new ScreenBroadcastListener(this);

listener.registerListener(new ScreenBroadcastListener.ScreenStateListener() {

@Override

public void onScreenOn() {

screenManager.finishActivity();

}

@Override

public void onScreenOff() {

screenManager.startActivity();

}

});

}

}

按下back之后，进行锁屏，现在测试一下oom\_adj的值

https://upload-images.jianshu.io/upload_images/1836169-c7bb89e3b3691357.png?imageMogr2/auto-orient/strip%7CimageView2/2/w/700

果然将进程的优先级提高了。

但是还有一个问题，内存也是一个考虑的因素，内存越多会被最先kill掉，所以把上面的业务逻辑放到Service中，而Service是在另外一个 进程中，在MainActivity开启这个服务就行了，这样这个进程就更加的轻量，

public class LiveService extends Service {

public static void toLiveService(Context pContext){

Intent intent=new Intent(pContext,LiveService.class);

pContext.startService(intent);

}

@Nullable

@Override

public IBinder onBind(Intent intent) {

return null;

}

@Override

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

//屏幕关闭的时候启动一个1像素的Activity，开屏的时候关闭Activity

final ScreenManager screenManager = ScreenManager.getInstance(LiveService.this);

ScreenBroadcastListener listener = new ScreenBroadcastListener(this);

listener.registerListener(new ScreenBroadcastListener.ScreenStateListener() {

@Override

public void onScreenOn() {

screenManager.finishActivity();

}

@Override

public void onScreenOff() {

screenManager.startActivity();

}

});

return START\_REDELIVER\_INTENT;

}

}

<service android:name=".LiveService"

android:process=":live\_service"/>

OK，通过上面的操作，我们的应用就始终和前台进程是一样的优先级了，为了省电，系统检测到锁屏事件后一段时间内会杀死后台进程，如果采取这种方案，就可以避免了这个问题。但是还是有被杀掉的可能，所以我们还需要做双进程守护，关于双进程守护，比较适合的就是aidl的那种方式，但是这个不是完全的靠谱，原理是A进程死的时候，B还在活着，B可以将A进程拉起来，反之，B进程死的时候，A还活着，A可以将B拉起来。所以双进程守护的前提是，系统杀进程只能一个个的去杀，如果一次性杀两个，这种方法也是不OK的。

事实上  
那么我们先来看看Android5.0以下的源码，ActivityManagerService是如何关闭在应用退出后清理内存的

Process.killProcessQuiet(pid);

应用退出后，ActivityManagerService就把主进程给杀死了，但是，在Android5.0以后，ActivityManagerService却是这样处理的：

Process.killProcessQuiet(app.pid);

Process.killProcessGroup(app.info.uid, app.pid);

在应用退出后，ActivityManagerService不仅把主进程给杀死，另外把主进程所属的进程组一并杀死，这样一来，由于子进程和主进程在同一进程组，子进程在做的事情，也就停止了。所以在Android5.0以后的手机应用在进程被杀死后，要采用其他方案。

2、前台服务

这种大部分人都了解，据说这个微信也用过的进程保活方案，移步[微信Android客户端后台保活经验分享](https://link.jianshu.com?t=http%3A%2F%2Fmt.sohu.com%2F20160408%2Fn443685317.shtml)，这方案实际利用了Android前台service的漏洞。  
原理如下  
对于 API level < 18 ：调用startForeground(ID， new Notification())，发送空的Notification ，图标则不会显示。  
对于 API level >= 18：在需要提优先级的service A启动一个InnerService，两个服务同时startForeground，且绑定同样的 ID。Stop 掉InnerService ，这样通知栏图标即被移除。

public class KeepLiveService extends Service {

public static final int NOTIFICATION\_ID=0x11;

public KeepLiveService() {

}

@Override

public IBinder onBind(Intent intent) {

throw new UnsupportedOperationException("Not yet implemented");

}

@Override

public void onCreate() {

super.onCreate();

//API 18以下，直接发送Notification并将其置为前台

if (Build.VERSION.SDK\_INT <Build.VERSION\_CODES.JELLY\_BEAN\_MR2) {

startForeground(NOTIFICATION\_ID, new Notification());

} else {

//API 18以上，发送Notification并将其置为前台后，启动InnerService

Notification.Builder builder = new Notification.Builder(this);

builder.setSmallIcon(R.mipmap.ic\_launcher);

startForeground(NOTIFICATION\_ID, builder.build());

startService(new Intent(this, InnerService.class));

}

}

public static class InnerService extends Service{

@Override

public IBinder onBind(Intent intent) {

return null;

}

@Override

public void onCreate() {

super.onCreate();

//发送与KeepLiveService中ID相同的Notification，然后将其取消并取消自己的前台显示

Notification.Builder builder = new Notification.Builder(this);

builder.setSmallIcon(R.mipmap.ic\_launcher);

startForeground(NOTIFICATION\_ID, builder.build());

new Handler().postDelayed(new Runnable() {

@Override

public void run() {

stopForeground(true);

NotificationManager manager = (NotificationManager) getSystemService(NOTIFICATION\_SERVICE);

manager.cancel(NOTIFICATION\_ID);

stopSelf();

}

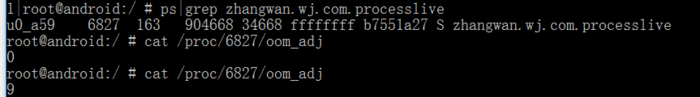
},100);

}

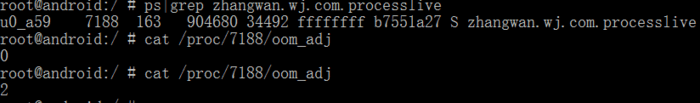
}

}

在没有采取前台服务之前，启动应用，oom\_adj值是0，按下返回键之后，变成9（不同ROM可能不一样）



在采取前台服务之后，启动应用，oom\_adj值是0，按下返回键之后，变成2（不同ROM可能不一样），确实进程的优先级有所提高。



3、相互唤醒

相互唤醒的意思就是，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。这个完全有可能的。此外，开机，网络切换、拍照、拍视频时候，利用系统产生的广播也能唤醒app，不过Android N已经将这三种广播取消了。



LBE安全大师



LBE安全大师

如果应用想保活，要是QQ，微信愿意救你也行，有多少手机上没有QQ，微信呢？或者像友盟，信鸽这种推送SDK，也存在唤醒app的功能。  
拉活方法

4、JobSheduler

JobSheduler是作为进程死后复活的一种手段，native进程方式最大缺点是费电， Native 进程费电的原因是感知主进程是否存活有两种实现方式，在 Native 进程中通过死循环或定时器，轮训判断主进程是否存活，当主进程不存活时进行拉活。其次5.0以上系统不支持。 但是JobSheduler可以替代在Android5.0以上native进程方式，这种方式即使用户强制关闭，也能被拉起来，亲测可行。

JobSheduler@TargetApi(Build.VERSION\_CODES.LOLLIPOP)

public class MyJobService extends JobService {

@Override

public void onCreate() {

super.onCreate();

startJobSheduler();

}

public void startJobSheduler() {

try {

JobInfo.Builder builder = new JobInfo.Builder(1, new ComponentName(getPackageName(), MyJobService.class.getName()));

builder.setPeriodic(5);

builder.setPersisted(true);

JobScheduler jobScheduler = (JobScheduler) this.getSystemService(Context.JOB\_SCHEDULER\_SERVICE);

jobScheduler.schedule(builder.build());

} catch (Exception ex) {

ex.printStackTrace();

}

}

@Override

public boolean onStartJob(JobParameters jobParameters) {

return false;

}

@Override

public boolean onStopJob(JobParameters jobParameters) {

return false;

}

}

5、粘性服务&与系统服务捆绑

这个是系统自带的，onStartCommand方法必须具有一个整形的返回值，这个整形的返回值用来告诉系统在服务启动完毕后，如果被Kill，系统将如何操作，这种方案虽然可以，但是在某些情况or某些定制ROM上可能失效，我认为可以多做一种保保守方案。

@Override

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

return START\_REDELIVER\_INTENT;

}

START\_STICKY  
如果系统在onStartCommand返回后被销毁，系统将会重新创建服务并依次调用onCreate和onStartCommand（注意：根据测试Android2.3.3以下版本只会调用onCreate根本不会调用onStartCommand，Android4.0可以办到），这种相当于服务又重新启动恢复到之前的状态了）。

START\_NOT\_STICKY  
如果系统在onStartCommand返回后被销毁，如果返回该值，则在执行完onStartCommand方法后如果Service被杀掉系统将不会重启该服务。

START\_REDELIVER\_INTENT  
START\_STICKY的兼容版本，不同的是其不保证服务被杀后一定能重启。

相比与粘性服务与系统服务捆绑更厉害一点，这个来自[爱哥](https://link.jianshu.com?t=http%3A%2F%2Fblog.csdn.net%2Faigestudio%2Farticle%2Fdetails%2F51348408)的研究，这里说的系统服务很好理解，比如NotificationListenerService，NotificationListenerService就是一个监听通知的服务，只要手机收到了通知，NotificationListenerService都能监听到，即时用户把进程杀死，也能重启，所以说要是把这个服务放到我们的进程之中，那么就可以呵呵了

@TargetApi(Build.VERSION\_CODES.JELLY\_BEAN\_MR2)

public class LiveService extends NotificationListenerService {

public LiveService() {

}

@Override

public void onNotificationPosted(StatusBarNotification sbn) {

}

@Override

public void onNotificationRemoved(StatusBarNotification sbn) {

}

}

但是这种方式需要权限

<service

android:name=".LiveService"

android:permission="android.permission.BIND\_NOTIFICATION\_LISTENER\_SERVICE">

<intent-filter>

<action android:name="android.service.notification.NotificationListenerService" />

</intent-filter>

</service>

所以你的应用要是有消息推送的话，那么可以用这种方式去欺骗用户。

结束：  
听说账号同步唤醒APP这种机制很不错，用户强制停止都杀不起创建一个账号并设置同步器，创建周期同步，系统会自动调用同步器，这样就能激活我们的APP，局限是国产机会修改最短同步周期（魅蓝NOTE2长达30分钟），并且需要联网才能使用。在国内各大ROM"欣欣向荣"的大背景下，关于进程保活，不加入白名单，我也很想知道有没有一个应用永活的方案，这种方案性能好，不费电，或许做不到，或许有牛人可以，但是，通过上面几种措施，在绝大部分的机型下，绝大部分用户手机中，我们的进程寿命确实得到了提高,这篇文章都是围绕怎么去提高进程oom\_adj的值来进行，关于降低内存占用方面，移步  
[Android性能优化的方方面面](https://www.jianshu.com/p/b3b09fa29f65)，所以关于我们的应用更加“健康”，性能优化必不可少。

[DEMO地址：https://github.com/herojing/KeepProcessLive](https://link.jianshu.com?t=https%3A%2F%2Fgithub.com%2Fherojing%2FKeepProcessLive)

作者：LooperJing  
链接：https://www.jianshu.com/p/1da4541b70ad  
來源：简书  
简书著作权归作者所有，任何形式的转载都请联系作者获得授权并注明出处。