robert's home

爱出者爱返 福往者福来

[置顶] Android service 不被杀死“永不退出的服务”（双进程，服务，多进程

标签： Android service 双进程 服务 服务不被杀掉

2016-12-21 20:24 4552人阅读 评论(2) 收藏 举报

分类：

Android（11）

关闭

本文的应用部分使用蓝色表示

介绍服务开发：

介绍关于服务不被杀死，以及微信的永久驻村谜底；

第一次做关于服务的开发，看了很多文章，终于有个大概，任后开始写程序测试并感受。在这里把学到的东西汇总一下，分享给大家。

随便一搜索很对关于Android开发service的文章，所以这简要说明。有一篇文章说的很好：“它是一种长生命周期的，没有可视化界面，运行于后台的一种服务程序。比如我们播放音乐的 时候，有可能想边听音乐边干些其他事情，当退出播放音乐的应用，如果不用Service，我 们就听不到歌了，所用到Service了。”

我喜欢这个解释。当然除此之外，普通应用使用了service使得整个应用有好的耦合性（功能依靠，相对比较独立）。就从字面意思来理解，提供服务的功能部分一般适合开发成一个服务。比如蓝牙连接服务。网络通信服务。音乐播放服务。当然我们自己开发的服务一般只给自己的应用。

以下简单汇总一下其他文章提到的精华部分，以及不全其他文章不足的地方

关闭

1．Android service分类：startservice bindservice两种方式。

startservice适合服务与应用程序在一块的情况，service和调用的应用程序在同一个进程里面。一般用于执行更新，加载等费事的操作。

Bindservice适用于完成一个独立功能的部分。不如一个蓝牙服务，需要蓝牙服务的app只需要bind服务就可以使用这个服务，相对于startservice来说，bindservice给更多调用者服务。

生命周期，网上文章说，startservice调用者不在了，还可以继续执行，

Ionic2之显示图表（chart）曲线.. (1488)
ionic2屏幕适配，适配手机，平.. (1368)

评论排行

Android service 不被杀死“永不… (2)
ViewPager+Fragment动态增加… (1)
安卓cordove插件开发指导(andr… (1)
android AppCompat, splash启动… (1)
ionic开发介绍之gulp文件内容… (0)
cordova工程webview注入本地j… (0)
ionic开发介绍之bower 介绍 (0)
ionic开发介绍之NPM管理工具… (0)
ionic开发介绍之环境搭建 (0)
plugin development guide（翻译.. (0)

最新评论

android AppCompat, splash启动白屏（黑屏…
Han_Peng ：第六点说的不对，如果设置了android:windowIsTranslucent属性，那么前面设置的...

Android service 不被杀死“永不退出的服务…
来自星星的谢广坤 ：按照楼主的说法现在有很多的APP都在小米、华为、OPPO、VIVO等等手机厂商那里进入了白名单？

安卓cordove插件开发指导(android plugin d…
孙华强 ：CSDN博友你好，我是孙华强，现将此篇博文收录进“Android知识库”。CSDN现在在做CSDN博...

ViewPager+Fragment动态增加页面，删除…
qq_32929105 ：不错，正好需要

Android service 不被杀死“永不退出的服务…

而，bindservice可以支持好多调用者绑定，当时一旦所有绑定者都死掉的化，bindservice也就退出了。当然startservice调用的时候会启动服务，bindservice在调用的时候如果服务还没有被启动则会启动服务。我在实际测试中，只要用清理工具清理了调用服务的软件，调用者不在了，无论那种方式启动的服务都会被关闭。

下面专门说一下“永不退出的服务”

为了本文方便描述：先介绍测试中用到的两种关闭服务的方法：

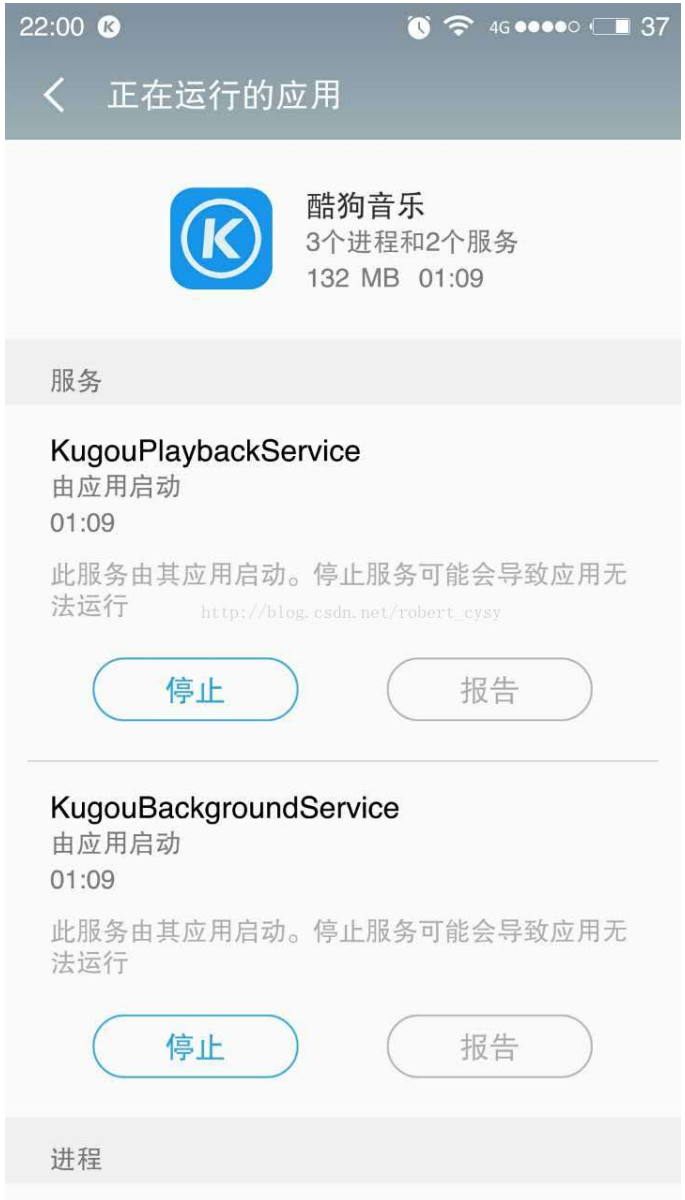
在用使用清理运行软件的功能作为关闭服务的第一种测试方法：向侧面滑动应用就被关闭运行的软件。

关闭

AM小可乐 ：很赞，之前项目里边想弄成微信这样的不死进程的，后来用的是双进程互相监听，但还是不行，在你这找到答案了...

在测试时关闭服务的第二种测试方式：在“正在运行的软件”里点击停止

关闭



关闭

关于周期网上有好多文章都是提到了“不死的服务”。很多文章提到了做出一个不死的服务。具体提到的方式有：

onStartCommand方法，返回START_STICKY

也就是在service的onstartcommand函数里返回这个值

```
@Override
public int onStartCommand(Intent intent, int flags, int startId) {
    flags = START_STICKY;
    return super.onStartCommand(intent, flags, startId);
}
```

这个时候如果清理的是一个设置了START_STICKY的app，用方法一和二的时候服务都会被

我测试多了，发现在清理软件里吧这个应用加入白名单之后。在使用方法一的时候。acriv

了，但是服务不会被清理。，使用方法二的时候，服务会被关闭，然后一段时间又会重新启动出现。即

START_STICKY产生的效果。

1．设置服务为前台进程。startForeground(0x111, notification);具体操作过程查看其他文章

效果，实现了前台进程其实就是在状态栏显示一个提示，应该是用来告诉用户所执行的动作在后台还在进行者，只能说service的优先级提高了，在系统内存不足的时候不会被第一个销毁。但是用户是可以通过方法一和二关闭这个服务的。（即调用者死掉，其服务也会被杀掉）。

在service的ondestroy里发送开启服务的intent。以及开启两个service相互监视，谁死掉就发送intent启动谁。本人没有测试。由于看清了服务的一些使用特性，觉得开发这样死活关不掉的应用实在是不好。

还有就是要做成系统应用，即把手机开启root账号，然后安装app到系统到system/app目录下。以使app获取系统级别的启动权限。

关闭

还有人为了达到不是服务真是用心良苦。居然研究了从linux底层开启进程做起，因为是看到像微信，搜狐。说实话，很佩服这些人的头脑以及技术。

以下这三篇文章讲的很好关于这方面。

http://blog.csdn.net/ztemt_sw2/article/details/27101681

http://download.csdn.net/detail/mihang2/9283985

http://blog.csdn.net/huaxun66/article/details/53158162

总结一下自己为啥放弃开发一个不死的服务。

关闭

1．需求方面：只有我的软件在运行的时候，我的服务才有存在的必要。

2．我用的是魅族和华为手机做的测试，使用清理后台应用（即：测试方法一）只要activity被杀死，那么跟着其对应的service也被杀死，对于网上来说的很多实现不被杀死方法都是因为其测试用的手机系统比较原生，一般不会强制杀死一个没有被调用的服务。当然我也没有在模拟器里运行测试，就是在两个手机上运行测试，是这样的。

3．发现微信也不是不死的app。发现酷狗音乐这种播放音乐的软件也会在activity死掉的时候，停止播放音乐。

4．关于用户体验的思考。对于普通应用没有必要在关闭activity之后仍然运行service。

关于要不要开启一个不死的服务：看到了一片英文文章，关于要不要做一个不死服务的必

好。

原文如下：

Diamonds Are Forever. ServicesA

By **AndroidGuys**

Android offers a “service” component. Unlike “activities”(screens) that interact directly with the user, services are more for managingbackground operations. The quintessential example is a music player, continuingto play tunes while its UI is not running.

Some developers then make the leap from “it is possible” to “itis a really good idea and should be used wherever”. Many people on AndroidGoogle Groups or StackOverflow propose to create services that “run forever”and ask how to prevent them from being killed off, and such.

关闭

These developers appear to ignore the costs of this approach:

· While running, a serviceconsumes one process' worth of memory, which is a substantial chunk, despiteDalvik's heavy use of copy-on-write techniques to share memory betweenprocesses. A single service hogging memory is not that big of a deal; a dozensuch services will prevent the Android device from being usable. Hence, even ifthe memory footprint does not impact the developer directly, it impacts theusers indirectly — much like how pollution benefits the polluter with acorresponding cost to society.

· While running, the servicewill need to fight Android, which will want to shut down the serv too tight. While Android should, in principle, restart the service oncememory is plentiful again, th will have no control over the timingof this.

· While running, the servicewill need to fight its own users, who may elect to shut down the service viathe Manage Applications screen, or through a third-party task manager. In thiscase, Android will not restart the service automatically.

· The service will fall asleepwhen the device falls asleep and shuts down the CPU, which means theservice "runs forever", it will not be able to *run* forever, unless it prevents the CPUfrom stopping, which wrecks battery life.

The recommended alternative is to use AlarmManager, as describedin a previous post and in finer Android programming books. Think of yourAndroid service as a cron job or Windows scheduled task, not as a persistentdaemon or Windows service. This approach works much better with Android'sframework and device limitations:

· The service only uses memorywhen it is actually doing something, not just sitting around waiting.

关闭

· The odds that Android willneed to kill off the service decreases, in part because the service will not be“old” and other services are likely to be killed first.

· The odds that the user willneed to kill off the service decreases, because the service is less likely tocause any pollution that may cause the user problems.

· The service can hold aWakeLock for the duration of its bit of work to keep the CPU running for ashort period

Now, the AlarmManager API is not the friendliest. Somedevelopers get tripped up while trying to multiple outstanding alarms.While there are ways to deal with this (via careful construction ofPen objects), sometimes you do not truly need more than one alarm. Ifyou have code that you want to minutes, and other code that youwant to run every 24 hours, use one 15-minute alarm and trigger theonce-every-24-hours code every 96th time the 15-minute alarm goes off.

I encourage Android developers to try to avoid long-runningservices wherever possible. If you are to design an app to avoidthe long-running service, post a clear description of the business scenario(level technical stuff) to the [android-developers] group or to StackOverflow with the #android tag. We can try to help you find ways ofachieving your business objectives in an Android-friendly fashion.

关闭

原文大体翻译如下：

很多开发者为了达到服务随时被启动起来的activity调用，而忽略了以下这些消耗：

在运行的时候，尽管dalvik使用了用时复制技术来共享相同的内存。但是一个存留的服务耗费了一个进程所占用的内存。

在运行的时候，RAM耗尽的时候，服务仍然对抗android系统不让杀掉自己

在运行的时候，服务要对抗关闭它的用户。（用户用应用管理器或第三方管理器）

在运行的时候，当设备睡眠的时候，CPU被关闭。即服务也被关闭了，除非这个服务开启阻挡手机睡眠选项。这就意味着电池电量的消耗。

因此推荐的方法是：使用AlarmManager。把你的服务当成定时执行任务或者计划执行任务。而不是一个永久的服务就像windows服务那样。

以下这样的设计方法会是的应用更好的与android系统构架和谐并存。

1. 只有服务真的在做事情的时候才开启服务，不要让服务坐着干等
2. 其实android杀死服务的几率很低。因为只要你正在使用服务，服务就不会变老。系统也就会杀掉其他服务而不是你正在使用的服务。
3. 用户停掉服务的几率也在减少。因为一个好的服务不会给用户造成影响，也不会给用户
4. 服务在其工作期间可以使用一段时间的WakeLock来保持CPU保持运作。但用完时候马上WakeLock。

我鼓励android开发者在可能的情况下尽量避免开发长时间运行的服务。

以上是原文精华。

那么说这样设计的例子有哪些。下面我举“酷狗音乐”的例子。

当你打开“酷狗音乐”并且播放了一首音乐，音乐在播放，你按下home键，然后activity被放置在后天，这个时候，后台服务来播放音乐。在播放的时候会在消息栏显示一个正在播放的音乐状态，以及控制按

关闭

钮，这个部分是把服务设置为前台服务。提高了服务的优先级。那么现在你使用我上面说的方法一，即把“酷狗音乐”的activity真正关闭之后，（调用者死掉）这个时候你会发现之前正在播放的音乐也会立即停止掉。即没有使用者服务就立马停掉。再进一步使用方法二手动关闭“酷狗音乐”的服务。音乐也会停止，并不再启动。这样的设计即使遵循了上面那篇英文文章所描述的方法。就是系统和用户其实在需要你服务的时候是不会把你的服务停掉的，除非系统和用户真的就是要停止你，那么你这个app为啥还不让停止的。

Android系统不会无端停止一个正在被使用的service。除非你这service自己就自己一个人硬抗不住的。无论你用什么手段。除非系统说你是个好人放过你吧。

下面介绍微信为什么死不掉。先说结果。微信其实也是遵循英文文章提到的设计模式。即你[illegible]聊天的时候，服务也一直在后台运行。一般用户关闭微信是使用home键。即activity放在了[illegible]信”中调用者的身份还在。

但是，但是接下来的就是神奇的部分，无论使用方法一还是方法二，都阻止不了微信继续可以收到消息。为啥呢，微信到底是使用了什么手段。什么黑科技。而且后台确实有两个进程，每个进程里都有一个或几个服务。而且用方法二关闭服务，很快服务又会重新出现。难道这个真是有两个服务互相监视，或者是有个非常底层的一个守护进程来保持应用不退出。

就在此时我没有写代码，而是下载了一个号称开启守护进程用jni开发的不死例子。

例子网址如下：http://download.csdn.net/detail/mihang2/9283985

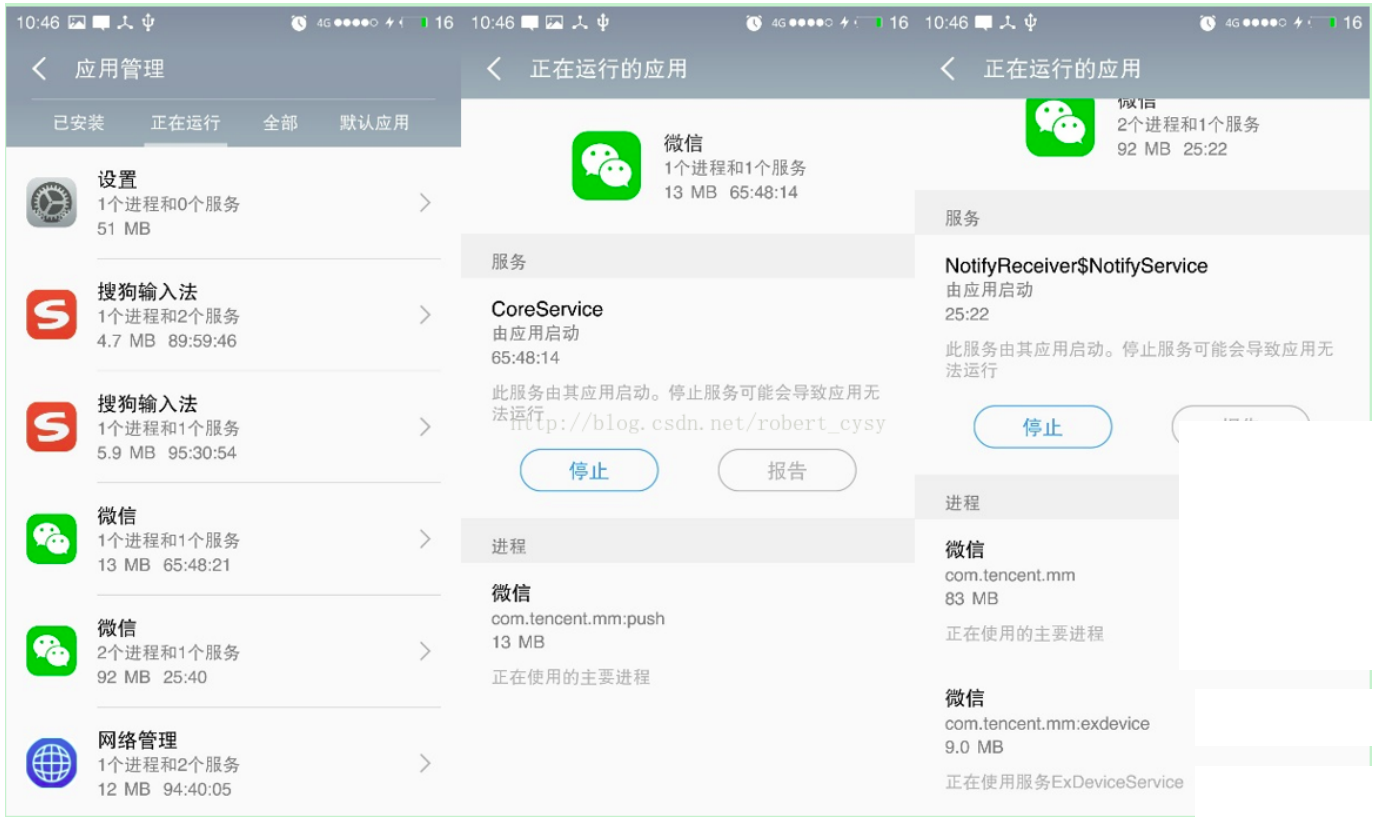
关闭

下载下来并安装到手机上，使用方法一个和二测试。方法二的时候，“正在运行的应用程序”界面确实可以显示停止了，即在界面里面看不到这个服务的身影，但是从调试信息里看到服务仍然还在运行者，虽然“正在运行的应用程序”里看不到服务的身影。然后使用方法一的测试结果是，服务马上就死掉了。那么回头继续来总结一下，为什么方法二没有真正停止服务呢，我想是因为虽然activity被放置在后台，它是它并没有死，还在调用服务。所以服务并没有被系统正真杀掉，系统知道调用者activity还在运行。那么为啥用方法一，服务马上就死掉了，因为系统发现你这activity不在了，留下你这个服务也没啥用，浪费资源和电池干脆把你立刻就停止。（当然这里只服务和调用activity在同一个进程里，即
程里，如果没有调用者，这个服务仍然会被杀掉）

那么接着回来看“微信”进程为啥可以不死。我们从另一个角度观察微信的进程。

首先从手机正在运行的程序来看是如下结果：

关闭



从上面的截图也可以看到这几个进程以及对应的服务

关闭

大家都是Android开发者，我是在windows下开发。打开命令提示符，插入一个开启了调试模式的手机，登录运行一个微信，输入："adb shell"这个时候你就进入了android手机的命令行终端，（linux终端）。可以执行linux命令。然后执行top命令查看当前运行的进程。执行如下命令:top –n 1 | greptencent

```
130|shell@m3note:/ $ top -n 1 | grep  tencent
 4087  0   0% S    28 1678872K  25176K  fg u0_a88   com.tencent.mm:exdevice
 4221  0   0% S    88 2030936K 124892K  fg u0_a88   com.tencent.mm
29839  0   0% S    28 1673420K  25896K  fg u0_a88   com.tencent.mm:push
shell@m3note:/ $
```

可以看到微信有三个进程，进程pid分别是“4087”，“4221”，“29839”

```
shell@m3note:/ $ ps
USER     PID   PPID  VSIZE  RSS     WCHAN    PC         NAME
root      1     0     17096  900   ffffffff 00000000 S /init
root      2     0     0      0     ffffffff 00000000 S kthreadd
root      3     2     0      0     ffffffff 00000000 S ksoftirqd/0
root      5     2     0      0     ffffffff 00000000 S kworker/0:0H
root      7     2     0      0     ffffffff 00000000 S migration/0
root      8     2     0      0     ffffffff 00000000 S rcu_preempt
root      9     2     0      0     ffffffff 00000000 S rcu_bh
shell@m3note:/ $ ps | grep tencent
u0_a88    4087  396   1678872 25744 ffffffff 00000000 S com.tencent.mm:exdevice
u0_a88    4221  396   2131944 180344 ffffffff 00000000 S com.tencent.mm
u0_a88    7699  396   2101724 95716 ffffffff 00000000 S com.tencent.mm:tools
u0_a88    29839 396   1673420 28792 ffffffff 00000000 S com.tencent.mm:push
shell@m3note:/ $ top -n 1 | grep  tencent
 4087  0   0% S    28 1678872K  25744K  fg u0_a88   com.tencent.mm:exdevice
 4221  0   0% S    90 2131944K 180344K  fg u0_a88   com.tencent.mm
 7699  0   0% S    74 2101724K  95716K  bg u0_a88   com.tencent.mm:tools
29839  0   0% S    28 1673420K  28792K  fg u0_a88   com.tencent.mm:push
```

可以从ps的结果发现，虽然是三个独立的进程，但是他们的父进程都是396，也就是说三个进程都是由同一个进场所创建。

关闭

继续往下看，现在查看我之前下载的那个关于使用的守护进程（使用jni开发的那个）例子http://download.csdn.net/detail/mihang2/9283985。

运行这个软件，然后用top命令和ps命令看：

```
shell@m3note:/ $ top -n 1 | grep  example
 9874  7   2% S     2 1584520K  19484K  fg u0_a162  com.example.dameonservice
 9846  0   0% S    33 1631420K  53936K  fg u0_a162  com.example.dameonservice
shell@m3note:/ $ ps | grep example
u0_a162   9846  396   1631420 53936 ffffffff 00000000 S com.example.dameonservice
u0_a162   9874  9846  1584520 19484 ffffffff 00000000 S com.example.dameonservice
```

发现问题了吗？，dameonsercice确实是有两个进程但是者两个进程的父进程居然不一样，而微信的是一样的。难道这就是其所说的守护进程。然后继续分析，看到另一个父进程不是396的进程的父进程为9846，而这个9846，是这个dameonservice进程的pid进程。也就是说。这个守护进程是有原来的进程创建而来的。，因此当我使用方法一关掉activity的时候，相当于把父进程销毁了，但是根据linux的知识，这个自进程不会被销毁，应该会被init进程收留成为正真linux下的守护进程，我查看了一下其它父进程为1的进程（linux守护进程）都是系统的软件，没有任何app是属于守护进程的。应该是Android构架限制了吧。所以这个作者做的守护进程方法，经过测试，其实是没有成功的。

先不管这个，，先看看为啥其中一个父进程和微信的父进程一样呢。所以来看一下，关于
程，首先根据linux，子进程都是被父进程创建（fork）而来的，也就是说微信和这个应用都有同一个父进程。使用ps | grep 396和 top –n 1 | grep396查看结果如下：

```
shell@m3note:/ $ ps | grep 396
root      396   1     1552544 11540 ffffffff 00000000 S zygote
u0_a84    1631  396   1579516 14140 ffffffff 00000000 S com.waves.maxxservice
u0_a53    1713  396   1609440 18536 ffffffff 00000000 S com.meizu.cloud
root      2396  2     0       0     ffffffff 00000000 S kbase_event
u0_a53    2832  396   1585076 20096 ffffffff 00000000 S com.meizu.c
u0_a88    4087  396   1678872 25460 ffffffff 00000000 S com.tencent.mm:exdevice
u0_a88    4221  396   2099360 159212 ffffffff 00000000 S com.tencent.mm
u0_a82    4341  396   1742716 45324 ffffffff 00000000 S com.sohu.inputmethod.sogou
u0_a82    5565  396   1610812 17408 ffffffff 00000000 S com.sohu.inputmethod.sogou:push_service
shell     7221  3446  18508  1040  000b396c a3be8e84 S /system/bin/sh
u0_a73    7461  396   1589968 25072 ffffffff 00000000 S com.meizu.flyme.weather
u0_a90    7505  396   1582992 26492 ffffffff 00000000 S com.meizu.net.pedometer
u0_a57    7988  396   1573088 24656 ffffffff 00000000 S com.meizu.flyme.service.find
u0_a162   9846  396   1631420 53936 ffffffff 00000000 S com.example.dameonservice
shell     9972  7221  20840  1396  00000000 b31b65fc R ps
shell     9973  7221  20840  1396  001a5664 84f425fc S grep
u0_a69    10944 396   1665616 28656 ffffffff 00000000 S com.meizu.net.search
u0_a82    13769 396   1607236 19716 ffffffff 00000000 S sogou.mobile.explorer.hotwords
u0_a32    19123 396   1696596 38672 ffffffff 00000000 S com.meizu.mstore
u0_a88    29839 396   1673420 27812 ffffffff 00000000 S com.tencent.mm:push
shell@m3note:/ $ top -n 1 | grep  396
  396  5   0% S     6 1552544K  11412K  fg root     zygote
 2396  5   0% S     1       0K      0K  fg root     kbase_event
shell@m3note:/ $
```

关闭

原来，所有运行的安卓app都是由zygote这个进程创建而来。仔细查看zygote的那一行。可以看到其pid即进程id为396。而其父进程为1，也就是说zygote是一个正真意义上的linux守护进程。又学习了一下。

最后这里给解释一下微信的这么多进程到底是怎么来的，由于以上测试可知，如果用jni用c++开启来开启一个进程的话，那么其父进程就不会是zygote。但是现在我们看到的微信所有父进程都是zygote也就是说微信没有使用jni里面创建进程。也就否定了其他博客对微信开启什么守护进程的猜想了。

那么微信的这些进程到底是怎么来的呢：

我是从这篇文章看到的相关信息

http://www.jcodecraeer.com/a/anzhuokaifa/androidkaifa/2015/0403/2686.html

文章提到了Androidmannifest.xml里面service部分的process元素。一下是一部分的原文：

理解Android进程

你应该已经知道了,安卓系统是基于Linux的。因此,每个应用程序都运行在其本身的进程(拥有一个独一无二的PID)中:这允许应用运行在一个相互隔离的环境中,不能被其他应用程序/进程干扰。通常来说,当你想启动一个应用程序,Android创建一个进程(从Zygote中fork出来的),并 创建一个主线程，然后开始运行Main Activity。

你可能不知道的是,你可以指定应用程序的一些组件运行在不同的进程中，而不是那个被用于启动应用程序的。先来看一下这个Nice的属性:

android:process

关闭

该进程属性可用于activities、services、content providers和broadcast receivers 和指定的进程中应该执行的特定组件。

在这个例子中,我指定MusicService必须执行在一个单独的“music”的进程:

```
<manifest ...>
  <application
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/Theme.Main" >
    <activity
      android:name=".MusicActivity"
      />
    <service
      android:name=".MusicService"
      android:process=":music"
    />
  </application>
</manifest>
```

然后我们看反编译微信里的Androidmannifest.xml，当然只反编译了资源文件。

```
<service android:name="com.tencent.mm.booter.CoreService" android:process=":push"/>
<service android:name="com.tencent.mm.booter.CoreService$InnerService" android:process=":push"/>
<service android:name="com.tencent.mm.booter.cache.CacheService" android:process=":push"/>
<service android:name="com.tencent.mm.plugin.hp.tinker.TinkerPatchResultService"/>
- <receiver android:name="com.tencent.mm.booter.MMReceivers$BootReceiver" android:process=":push">
  - <intent-filter>
      <action android:name="android.intent.action.BOOT_COMPLETED"/>
    </intent-filter>
</receiver>
- <receiver android:name="com.tencent.mm.booter.MMReceivers$ConnectionReceiver" android:process=":push">
  - <intent-filter>
      <action android:name="android.net.conn.CONNECTIVITY_CHANGE"/>
    </intent-filter>
</receiver>
```



```
shell@m3note:/ $ ps | grep tencent
u0_a88    4087  396   1678772 25468 ffffffff 00000000 S com.tencent.mm:exdevic
u0_a88    4221  396   2164880 162604 ffffffff 00000000 S com.tencent.mm
u0_a88    29839 396   1673420 28748 ffffffff 00000000 S com.tencent.mm:push
```


从内容可以看到其中我们在之前看到的push进程是怎样来的。

也就证明出了，微信开启进程的方法。

接下来看看“微信”在面对两种测试方法其进程到底是怎样的。（真正的揭秘微信不死的面纱

先从adb shell 命令里执行top –n 1 | grep tencent来查看一下微信正常运行是的状态

```
shell@m3note:/ $ top -n 1 | grep tencent
 4087  1   0% S    28 1678732K  24416K  fg u0_a88   com.tencent.mm:exdevice
 4221  0   0% S    97 2161588K 164104K  fg u0_a88   com.tencent.mm
29839  0   0% S    28 1673420K  29286K  fg u0_a88   com.tencent.mm:push
shell@m3note:/ $
```


现在使用方法二来停止服务：

我把“正在运行”里面的可以看到的服务都关闭了，然后运行top –n 1 | grep tencent结果如下：

```
shell@m3note:/ $ top -n 1 | grep tencent
 4221  0   0% S    97 2161612K 148372K  bg u0_a88   com.tencent.mm
 4087  0   0% S    28 1678782K  23044K  bg u0_a88   com.tencent.mm:exdevice
29839  0   0% S    28 1673420K  28180K  fg u0_a88   com.tencent.mm:push
```


关闭

可见服务并没有变化，当然按我的理解是，因为有一个后台的activity正在使用服务，所以服务没有被系统销毁了，因为我们自己的测试服务如果被方法二关闭的化，我们看到调试信息里，服务仍然在运行，也就是就像我们现在看到的，其实服务进程并没有被销毁。

然后使用方法一来停止服务：

结果是，这几条服务还在，这也就是人们感觉对微信的深不可测，没有被销毁。虽然，我后台的activity销毁了，但是服务没停。那么接下来我是怎么样发现微信的不死秘密的呢。

答案如下：

关闭

关闭

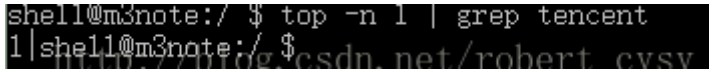
我以我的魅族手机为例：可以看到加速的时候（也就是我的测试方式一）其实并没有清除微信的任何进程，右面是华为的，华为的没有找到具体显示吧微信放在白名单的。但是很显然华为系统是有一个“内存加速白名单”，微信这样的应用就在白名单里。

再看下面俩张图。显示了魅族和华为的加速白名单。微信默认就在里面。

关闭

那么按照我看到这个之后的猜想，把微信从白名单里去掉或者把我自己开发的软件放到白名单都可以验证“微信”杀不掉的原因。

现在我吧微信从白名单里移除，然后使用方法一来测试：



可以看到，微信的进程都没有了，当然是死掉了。

知道微信为啥不死了吧，其实我查找文章的时候，看到一个论坛提到了说“系统给微信设置了白名单，其实微信没有做什么黑科技”，我当时感觉不信，就此折腾了几天。

当然，微信有好多服务和进程，只有当服务需要执行功能的时候，我们才可以看到服务进

微信把暂时不用的服务都停止掉了。我的测试不能排除微信使用了双服务互相启动，虽然

名单里。

当然可以从Androidmannifest.xml看出微信使用了开机启动，和激活屏幕启动。当然从华为

可以看到微信使用wakelock，就是微信在关闭屏幕之后还在后天运行。

关于wakelock详细内容查看以下文章。

http://blog.sina.com.cn/s/blog_4ad7c2540101n2k2.html

其中有个选项为：PARTIAL_WAKE_LOCK:保持CPU 运转，屏幕和键盘灯有可能是关闭的。

关闭

顶 3 踩 0

- 上一篇 Android制作service以及sdk并打包成jar包
- 下一篇 Cordova插件编写之调用service（jar打包的服务）

相关文章推荐

- 保持Service不被Kill掉的方法--双Service守护 && A…
- Presto服务治理与架构优化在京东的实践应用--王哲…
- Android SERVICE后台服务进程的守护
- 【免费直播】Python最佳学习路线--韦玮
- 关于Android Service真正的完全详解，你需要知道…
- JS-SDK开发与微信支付
- android service和activity跨进程通讯
- Spring Cloud微服务真实场景实战解析
- ionic开发——禁止手机自动旋转下横屏处
- 10小时深入掌握 Kubernetes
- Ionic2 隐藏状态栏，全屏， 禁止旋转，保
- JDK9新特性
- cordova app强制横屏
- 关于android 0进程1服务的意思
- Android进程级别与如何防止服务进程被回收
- Android开发之多进程详解

关闭

查看评论

来自星星的谢广坤 2楼 2017-08-28 17:15发表

按照楼主的说法现在有很多的APP都在小米、华为、OPPO、VIVO等等手机厂商那里进入了白名单？

AM小可乐 1楼 2016-12-22 09:24发表

很赞，之前项目里边想弄成微信这样的不死进程的，后来用的是双进程
互相监听，但还是不行，在你这找到答案了666