

# 什么情况需要使用多进程

## 常驻后台任务应用

类似音乐类、跑步健身类、手机管家类等长时间需要在后台运行的应用。这些应用的特点就是，当用户切到别的应用，或者关掉手机屏幕的时候，应用本身的核心模块还在正常运行，提供服务。如果因为手机内存过低，或者是进程重要性降低，导致应用被杀掉，后台服务停止，对于这些应用来说，就是灭顶之灾。合理利用多进程，将核心后台服务模块和其他UI模块进行分离，保证应用能更稳定的提供服务，从而提升用户体验。

举个例子：

现在要做一款音乐播放器，现在有以下几种方案：

- A. 在Activity中直接播放音乐。
- B. 启动后台Service，播放音乐。
- C. 启动前台Service，播放音乐。
- D. 在新的进程中，启动后台Service，播放音乐。
- E. 在新的进程中，启动前台Service，播放音乐。

首先我们分析A

在A中，我们的播放器是直接activity中启动的。首先这么做肯定是不对的，我们需要在后台播放音乐，所以当activity退出后就播不了了，之所以给出这个例子是为了控制变量作对比。

然后我们来看下A的使用场景。

音乐播放器无非是打开app，选歌，播放，退到桌面，切其他应用。我们选取了三个场景，打开、按home切换其他应用、按back退回桌面。让我们看一下A的相对应的oom\_adj、oom\_score、oom\_score\_adj的值。（下面三张图依次对应为【打开状态】、【按了Home键被切换状态】、【按了Back键被退出状态】）

```
root@libra:/ # cat /proc/21163/oom_adj
0
root@libra:/ # cat /proc/21163/oom_score
28
root@libra:/ # cat /proc/21163/oom_score_adj
0
```

上图为打开状态下oom\_adj、oom\_score、oom\_score\_adj的值

```
root@libra:/ # cat /proc/21163/oom_adj
7
root@libra:/ # cat /proc/21163/oom_score
438
root@libra:/ # cat /proc/21163/oom_score_adj
411
```

上图为按了Home键状态下oom\_adj、oom\_score、oom\_score\_adj的值

```
root@libra:/ # cat /proc/21163/oom_adj
9
root@libra:/ # cat /proc/21163/oom_score
673
root@libra:/ # cat /proc/21163/oom_score_adj
647
```

上图为按了Back键状态下oom\_adj、oom\_score、oom\_score\_adj的值

当我们应用在前台的时候，无论adj还是score还是score\_adj，他们的值都非常的小，基本不会被LMK所杀掉，但是当我们按了Home之后，进程的adj就会急剧增大，变为7，相应的score和score\_adj也会增大。在上篇文章中我们得知，adj=7即为被切换的进程，两个进程来回切换，上一个进程就会被设为7。当我们按Back键的时候，adj就会被设为9，也就是缓存进程，优先级比较低，有很大的几率被杀掉。

接着我们分析B

B是直接启动一个后台service并且播放音乐，这个处理看起来比A好了很多，那么实际上，B的各个场景的优先级和A又有什么不同呢？让我们来看下B的对应的打开、切换、退出相应的adj、score、score\_adj的值。（下面三张图依次对应为【打开状态】、【按了Home键被切换状态】、【按了Back键被退出状态】）

```
root@libra:/ # cat /proc/27615/oom_adj
0
root@libra:/ # cat /proc/27615/oom_score
28
root@libra:/ # cat /proc/27615/oom_score_adj
0
```

上图为打开状态下oom\_adj、oom\_score、oom\_score\_adj的值

```
root@libra:/ # cat /proc/27615/oom_adj
7
root@libra:/ # cat /proc/27615/oom_score
438
root@libra:/ # cat /proc/27615/oom_score_adj
411
```

上图为按了Home键状态下oom\_adj、oom\_score、oom\_score\_adj的值

```
root@libra:/ # cat /proc/27615/oom_adj
9
root@libra:/ # cat /proc/27615/oom_score
556
root@libra:/ # cat /proc/27615/oom_score_adj
647
```

上图为按了Back键状态下oom\_adj、oom\_score、oom\_score\_adj的值

B的情况其实是与A类似的，三种状态的adj、score\_adj的值都是一样的，只有score有一点出入，其实分析源码得知，LMK杀进程的时候，score的左右其实并不大，所以我们暂时忽略它。所以，与A相比，他们的adj和score\_adj的值都相同，如果遇到内存不足的情况下，这两个应用谁占得内存更大，谁就会被杀掉。不过鉴于A实在activity中播放音乐，所以B还是比A略好的方案。

这里有朋友肯定要问了，为什么切到后台后，adj的值是7而不是5，后台不是还有service在跑吗？

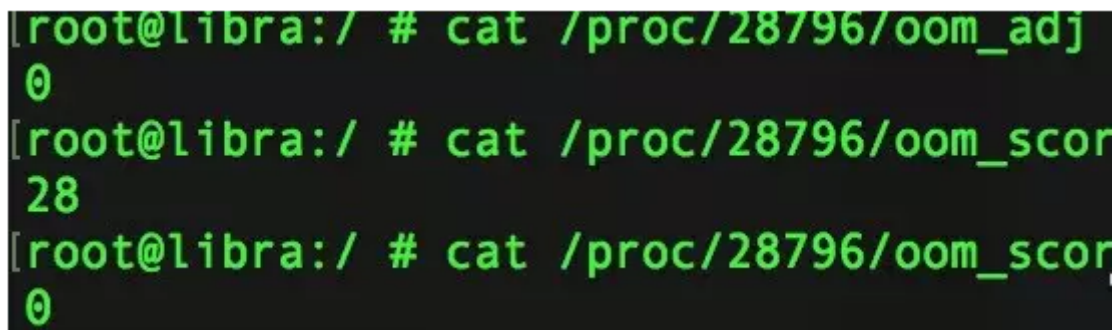
我们通过查看源码可以找出来，当切换Home的时候，会调用 `ActivityStack.java` 的 `finishCurrentActivityLocked` 函数，然后调用到了 `ActivityManagerService.java` 的 `computeOomAdjLocked` 函数，在这里，对进程的ADJ值进行重新计算。

```
if (app == mPreviousProcess && app.activities.size() > 0) {
    if (adj > ProcessList.PREVIOUS_APP_ADJ) {
        adj = ProcessList.PREVIOUS_APP_ADJ;
        schedGroup = Process.THREAD_GROUP_BG_NONINTERACTIVE;
        app.cached = false;
        app.adjType = "previous";
    }
    if (procState > ActivityManager.PROCESS_STATE_LAST_ACTIVITY) {
        procState = ActivityManager.PROCESS_STATE_LAST_ACTIVITY;
    }
}
```

当进程为PreviousProcess情况，则ADJ=7。

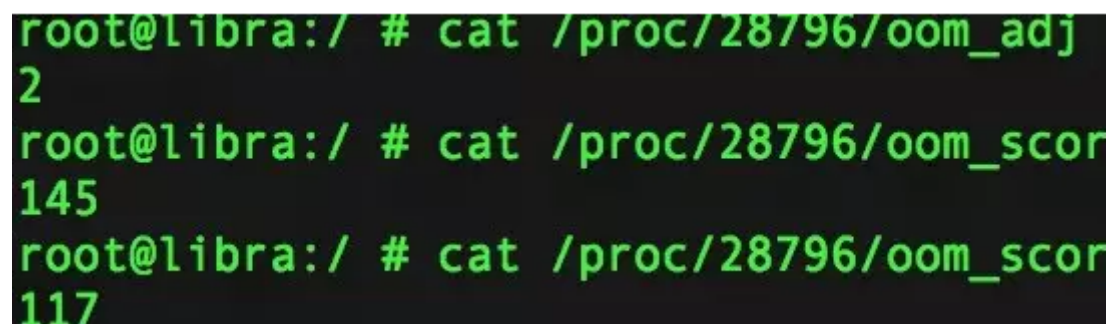
接下来分析C

C的话是启动一个前台Service来播放音乐。让我们来看一下对应的值。（下面三张图依次对应为【打开状态】、【按了Home键被切换状态】、【按了Back键被退出状态】）



```
[root@libra:/ # cat /proc/28796/oom_adj
0
[root@libra:/ # cat /proc/28796/oom_score
28
[root@libra:/ # cat /proc/28796/oom_score_adj
0
```

上图为打开状态下oom\_adj、oom\_score、oom\_score\_adj的值



```
root@libra:/ # cat /proc/28796/oom_adj
2
root@libra:/ # cat /proc/28796/oom_score
145
root@libra:/ # cat /proc/28796/oom_score_adj
117
```

上图为按了Home键状态下oom\_adj、oom\_score、oom\_score\_adj的值



```
root@libra:/ # cat /proc/28796/oom_adj
2
root@libra:/ # cat /proc/28796/oom_score
145
root@libra:/ # cat /proc/28796/oom_score_adj
117
```

上图为按了Back键状态下oom\_adj、oom\_score、oom\_score\_adj的值

在前台的时候，和AB是一样的，adj都是0，当切到后台，或者back结束时，C对应的adj就是2，也就是可感知进程。adj=2可以说是很高优先级了，非root手机，非系统应用已经没有办法将其杀掉了。

总的来说，C方案比B优秀，拥有前台Service的C更不容易被系统或者其他应用所杀掉了，进程的优先级一下子提高到了2，相对于B来说更稳定，用户体验更好。不过有一点不足是必须启动一个前台service。不过现在大部分的音乐类软件都会提供一个前台service，也就不是什么缺点了。其实也是有灰色方法可以启动一个不显示通知的前台service，这里就不过多介绍了。

那么还有可改进的余地吗？

答案当然是肯定的。

让我们来看看D，终于我们的主角，多进程登场了。

D把应用进行了拆分，把用于播放音乐的service放到了新的进程内，让我们看一下对应的值。（下面三张图依次对应为【打开状态】、【按了Home键被切换状态】、【按了Back键被退出状态】）

上图为打开状态下oom\_adj、oom\_score、oom\_score\_adj的值

```
root@libra:/ # cat /proc/5588/oom_adj
0
root@libra:/ # cat /proc/5588/oom_score
28
root@libra:/ # cat /proc/5588/oom_score_adj
0
```

上图为按了Home键状态下oom\_adj、oom\_score、oom\_score\_adj的值

```
root@libra:/ # cat /proc/5588/oom_adj
7
root@libra:/ # cat /proc/5588/oom_score
438
root@libra:/ # cat /proc/5588/oom_score_adj
411
```

上图为按了Back键状态下oom\_adj、oom\_score、oom\_score\_adj的值

```
root@libra:/ # cat /proc/5588/oom_adj
9
root@libra:/ # cat /proc/5588/oom_score
555
root@libra:/ # cat /proc/5588/oom_score_adj
529
```

上面三张图对应的是D应用主进程的ADJ相关值，我们可以看出来，跟A类似，adj都是0，7，9。由于少了service部分，内存使用变少，最后计算出的oom\_score\_adj也更低了，意味着主进程部分也更容易被杀死。

下面我们看下拆分出的service的相关值

```
[root@libra:/ # cat /proc/5938/oom_adj
5
[root@libra:/ # cat /proc/5938/oom_score
311
[root@libra:/ # cat /proc/5938/oom_score_adj
294
```

上图为后台Service的oom\_adj、oom\_score、oom\_score\_adj的值

因为是service进程，所以不受打开，关闭，切换所影响，这里就放了一张图。

我们可以看到，service的adj值一直是5，也就是活跃的服务进程，相比于B来说，优先级高了不少。不过对于C来说，其实这个方案反倒不如C的adj=2的前台进程更稳定。但是D可以自主释放主进程，使D实际所占用的内存很小，从而不容易被杀掉。那么到底C和D谁是更优秀的设计？我个人认为，在ABCDE这5个设计中，D是最具智慧的设计，具体是为什么？先卖个关子，等我们说完了E，再作总结。

那就赶紧分析E吧

E也是使用了多进程，并且在新进程中，使用了前台service，先看下对应的值。（下面三张图依次对应为【打开状态】、【按了Home键被切换状态】、【按了Back键被退出状态】）

```
[root@libra:/ # cat /proc/20468/oom_adj
0
[root@libra:/ # cat /proc/20468/oom_score
28
[root@libra:/ # cat /proc/20468/oom_score_adj
0
```

上图为打开状态下oom\_adj、oom\_score、oom\_score\_adj的值

```
root@libra:/ # cat /proc/20468/oom_adj
7
root@libra:/ # cat /proc/20468/oom_score
438
root@libra:/ # cat /proc/20468/oom_score_adj
411
```

上图为按了Home键状态下oom\_adj、oom\_score、oom\_score\_adj的值

```
root@libra:/ # cat /proc/20468/oom_adj
9
root@libra:/ # cat /proc/20468/oom_score
555
root@libra:/ # cat /proc/20468/oom_score_adj
529
```

上图为按了Back键状态下oom\_adj、oom\_score、oom\_score\_adj的值

```
root@libra:/ # cat /proc/32193/oom_adj
2
root@libra:/ # cat /proc/32193/oom_score
135
root@libra:/ # cat /proc/32193/oom_score_adj
117
```

这个不多解释，和ABD基本差不多，都是0，7，9。我们看下拆分出来的进程的值。

上图为后台Service的oom\_adj、oom\_score、oom\_score\_adj的值

我们可以看到，这个进程的值是2，像C方案，非常小，非常稳定，而且，我们还可以在系统进入后台后，手动杀掉主进程，使整个应用的内存消耗降到最低，内存低，优先级又高，E获得了今天的最稳定的方案奖。

## 小结

ABCDE，5种方案都已经分析完了。显然，E是最稳定的方案，不过，我刚才说过，我个人最倾向于D方案，并且认为D是最智慧的方案，这是为什么呢？

其实我们可以做个比喻，把整个Android系统比喻成一个旅游景点，Low Memory Killer就是景点的门卫兼保安，然后我们每个进程的ADJ相当于手里的门票，有的人是VIP门票，有的人是普通门票。景点平常没人的时候还好，谁拿票都能进，当人逐渐拥挤的时候，保安就开始根据票的等级，往外轰人。E方案就是一个拿着普通票的妈妈，带着一个VIP的孩子去参观，D方案就是一个拿着普通票的妈妈，带着一个拿着中等票的孩子参观。当内存不够的时候，保安会先把两个妈妈轰出去，孩子们在里面看，再不够了，就会把D孩子给轰出去。这么看来，显然E的效果更好一些，不过由于Android系统对于VIP票的发放没有节制，大家都可以领VIP票，那也就是相当于没有VIP票了。所以如果E方案是一种精明，那么D才是真正的智慧。将调度权还给系统，做好自己，维护好整个Android生态。

其实现阶段，如果公司要做一个后台类型的应用，我个人也是会选择追逐眼前利益，采用E方案的，这也是不得已而为之，大家都利用漏洞拿VIP票，你不拿，将来做出来的APP出了偏差，你是要负责的，所以还是希望Android能把漏洞堵住，把内存分配给真正需要的人，而我们自己也应该遵守规矩，维护整个生态平衡。



还有一点，是因为现在部分Root的手机都有优化大师，其实这个优化大师，就好比是个临时工门卫，告诉你他能解决景区爆满问题，实际上他的做法是，把一些票的等级降低，比如把中等票变成赠票，然后给你名正言顺的轰出去，听着是不是很耳熟？“让一部分人先富裕起来，然后把不富裕的杀掉，达成共同富裕。”

我的测试机之前装了某款优化软件，然后，在正常手机上的adj的值，都有一定程度的降低，来我们上证据。

```
root@libra:/ # cat /proc/30572/oom_adj
8
root@libra:/ # cat /proc/30572/oom_score
487
root@libra:/ # cat /proc/30572/oom_score_adj
470
```

上图为D方案下，Service进程的oom\_adj、oom\_score、oom\_score\_adj的值

看到没，安装了优化应用之后，本应该adj=5的活跃服务进程，被调整为8，意思是不活跃的服务进程，这种做法本身就违反了最初Android设计的思想。

还有

```
root@libra:/ # cat /proc/31686/oom_adj
11
root@libra:/ # cat /proc/31686/oom_score
673
root@libra:/ # cat /proc/31686/oom_score_adj
647
```

上图为E方案下，主进程在按了back键退出之后进程的oom\_adj、oom\_score、oom\_score\_adj的值

本来应该adj=9的缓存进程，调整为adj=11。adj=11在Android中都没给出定义。

所以，选择E也是无奈之举。还是呼吁大家要克制吧，维护Android的生态系统是每个工程师的责任。

“喜欢是放肆，但爱是克制” — 阿尔伯特·爱因斯坦 [手动滑稽]

## 多模块应用

多进程还有一种非常有用的场景，就是多模块应用。比如我做的应用大而全，里面肯定会有很多模块，假如有地图模块、大图浏览、自定义WebView等等（这些都是吃内存大户），还会有一些诸如下载服务，监控服务等等，一个成熟的应用一定是多模块化的。

首先多进程开发能为应用解决了OOM问题，Android对内存的限制是针对于进程的，这个阈值可以是48M、24M、16M等，视机型而定，所以，当我们需要加载大图之类的操作，可以在新的进程中去执行，避免主进程OOM。

多进程不光解决OOM问题，还能更有效、合理的利用内存。我们可以在适当的时候生成新的进程，在不需要的时候及时杀掉，合理分配，提升用户体验。减少系统被杀掉的风险。

多进程还能带来一个好处就是，单一进程崩溃并不影响整体应用的使用。例如我在图片浏览进程打开了一个过大的图片，java heap 申请内存失败，但是不影响我主进程的使用，而且，还能通过监控进程，将这个错误上报给系统，告知他在什么机型、环境下、产生了什么样的Bug，提升用户体验。

再一个好处就是，当我们的应用开发越来越大，模块越来越多，团队规模也越来越大，协作开发也是个很麻烦的事情。项目解耦，模块化，是这阶段的目标。通过模块解耦，开辟新的进程，独立的JVM，来达到数据解耦目的。模块之间互不干预，团队并行开发，责任分工也明确。至于模块化开发与多进程的结合，后续会写一篇专门的文章来研究这个问题。