


# Android N:开发者应注意什么？



作者

逍遥wqy (/u/c71b6a60e9f8)

+ 关注

2016.04.17 19:07 字数 2482 阅读 2352 评论 1 喜欢 7

(/u/c71b6a60e9f8)

Android 6.0这个棉花糖可能很多人还没尝到呢，但不管怎样，Android N还是要来了。

## Android N发布时间线

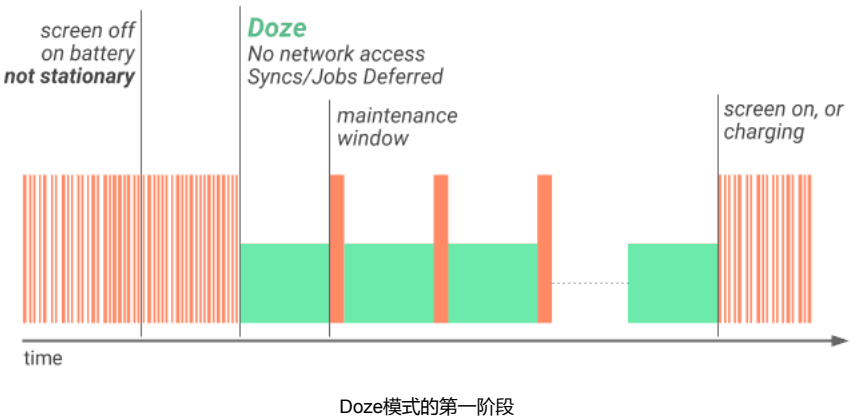
新的Android系统带来了很多新特性：

- 多窗口支持
  - 通知栏直接回复
  - 通知分组
  - Doze模式2.0

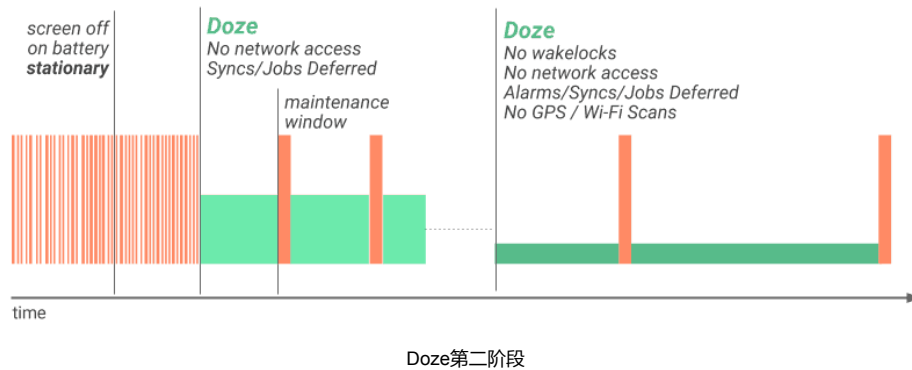
在关注这些新特性的同时，作为开发者，我们更应该注意新的是Android N给开发者带来了哪些改变？本文就给开发者朋友们讲述下我们应该注意什么（本文主要内容译自 Behavior Changes (<http://developer.android.com/intl/zh-cn/preview/behavior-changes.html>)一文，但还有很多笔者自己的理解）。

## Doze

其实Doze模式在6.0就引入了，在Android 6.0系统的手机上，如果用户拔掉电源，在关闭手机屏幕并且不动的一段时间后，系统便会进入Doze模式。此时，手机通过延缓CPU和网络活动减少电量的消耗。在Android N上，Doze模式的功能更加强大，在你拔掉电源关闭手机屏幕后，即使你的手机在“运动”，比如在你的口袋里随你一起摇摆，它也会进入Doze模式。



Doze模式的第一阶段：手机在非充电状态下锁屏一定时间后，Doze模式就开始工作了，它控制手机不再让你的app访问网络，推迟异步和同步任务的执行。如果手机在维持第一阶段一定时间后，Doze就升级到第二阶段，此时系统开始限制GPS使用、Wi-Fi扫描甚至还有 WakeLock 以及 AlarmManager 。



不过在Doze模式下手机也并非完全不工作，在特定的维护窗口期（ maintenance windows ），不管是在Doze的第一阶段还是第二阶段，系统将允许app访问网络并执行被推迟的异步或同步任务，但是这个窗口期很短暂。

注意，手机在屏幕点亮或者是插上电源时就会退出Doze模式，当然之前的限制也就不复存在了。那Doze模式中，app如何工作呢，假如我们的app是即时通信的应用，你把我网断了，那消息如何及时送达？针对该问题，Google给了解决方案：那就是集成GCM，GCM针对Doze和stand by模式做了优化，即使手机处于这两种模式下，GCM仍然可以保证消息到达，这样你就可以激活App了。但是，天朝的开发者的怎么办，GCM在我大天朝完全不靠谱呀！这时，作为开发者，我们就不得不考虑其他方式了，比如在自身的推送上增加第三方推送，Github上有个关于第三方推送集成的issue (<https://github.com/android-cn/topics/issues/4>)，有兴趣的可以看下。通过讨论区的留言可以看出大家统一的方案便是自身Push通道+第三方推送+小米Push+华为Push。在我看来小米Push必须要接入的，有两个原因：

- 1、小米手机用户量很大（毕竟国内出货量第一呀！）；
- 2、小米的一键清理太强大，可以杀掉所有app相关的进程（就算你是前台Service也是照杀不误）。

为了保证小米手机上用户能够及时收到消息，它自家的Push必须要接入呀，就算以后7.0 Doze 2.0闪亮登场，小米Push应该也能保证推送消息在小米手机上的正常到达吧。同样，基于同样的原因，华为Push也要考虑集成，这样至少可以保证华为手机用户可以及时收到消息了（不知道是不是我打开的姿势不对，华为Push的Demo在我的三星手机上通知栏推送就是无法弹出通知栏）。所以针对小米Push和华为Push，开发者在集成时还是要考虑到Rom的过滤，比如只在MIUI上开启小米Push，而只有在华为的EMUI上才打开华为Push，其它手机厂商还是以自身Push通道为主。哎，我天朝的开发者的日子就是悲催呀，不知道是不是以后三星、联想、魅族、HTC等等一众手机厂商都会给出自家的Push方案，或者针对Android的Doze给出解决方案，否则开发者的日子还怎么过呀.....

## 工程瘦身：后台优化

为了优化内存使用和电量的消耗，Android N去掉了三个隐式的广播。这个改变对用户来说绝对一大利好，因为后台注册这些广播的app在后台会经常被拉起，自然而然会影响到设备性能进而影响用户体验（很不幸，对开发者来讲，又有一些trick被限制了）。比如

CONNECTIVITY\_ACTION

([http://developer.android.com/reference/android/net/ConnectivityManager.html#CONNECTIVITY\\_ACTION](http://developer.android.com/reference/android/net/ConnectivityManager.html#CONNECTIVITY_ACTION))，在N之前的系统中，注册该广播的app在网络连接有变动时都会收到系统发出的广播，这样主进程被kill的app就可以复活了。此外还有 ACTION\_NEW\_PICTURE ([http://developer.android.com/reference/android/hardware/Camera.html#ACTION\\_NEW\\_PICTURE](http://developer.android.com/reference/android/hardware/Camera.html#ACTION_NEW_PICTURE))

和 ACTION\_NEW\_VIDEO

([http://developer.android.com/reference/android/hardware/Camera.html#ACTION\\_NEW\\_VIDEO](http://developer.android.com/reference/android/hardware/Camera.html#ACTION_NEW_VIDEO))。

对于这三个广播，Android N具体做了如下的优化：



- `CONNECTIVITY_ACTION` : `targetSdk`为Android N的app如果在后台就无法收到该广播,即使你在manifest中做了相应配置,但如果app在前台,依然还是可以收到。
- `ACTION_NEW_PICTURE` 和 `CONNECTIVITY_ACTION` : 这两个广播的优化会影响所有app,只要你的app运行在Android N系统的手机上,不管`targetSdk`是不是Android N,都会受到限制。

## 权限变更

Android N权限变化主要在于文件系统权限的更改,此外还有一个权限被废弃--

`GET_ACCOUNTS` ,在`targetSdk`为N的app中,系统将忽略 `GET_ACCOUNTS` 的请求,这里主要说下文件系统权限。

## 文件系统权限的变化

为了提高私有文件的安全性,在`targetSdk`版本为N或者以后版本的app中,其私有目录将会限制访问。这可以防止私有文件元数据的泄露,比如文件大小或者是文件是否存在。但这给开发者带来了很多不利的影响:

- 文件的owner不能放宽文件权限,如果你使用 `MODE_WORLD_READABLE` ([http://developer.android.com/reference/android/content/Context.html#MODE\\_WORLD\\_READABLE](http://developer.android.com/reference/android/content/Context.html#MODE_WORLD_READABLE)) 或者 `MODE_WORLD_WRITEABLE` ([http://developer.android.com/reference/android/content/Context.html#MODE\\_WORLD\\_WRITEABLE](http://developer.android.com/reference/android/content/Context.html#MODE_WORLD_WRITEABLE))操作文件,将会触发 `SecurityException` (<http://developer.android.com/reference/java/lang/SecurityException.html>)。
- 当你跨package域传递 `file://` 的URI时,接收者得到的将是一个无权访问的路径,因此,这将会触发 `FileUriExposedException`。对于这类操作,官方推荐的方式是使用 `FileProvider` (<http://developer.android.com/intl/zh-cn/reference/android/support/v4/content/FileProvider.html>),当然你也可以使用 `ContentProvider`。

这里只看文字理解起来可能有点小困难,所以我将以调用系统拍照为例说明下:在`targetSdk`为Android N之前的系统版本时,你可以使用如下方法调用系统相机拍照并存入指定路径中。

```
Intent intent = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
Uri uri = Uri.fromFile(sdcardsTempFile);
intent.putExtra(MediaStore.EXTRA_OUTPUT, uri);
```

但是当你将`targetSdk`设置为 Android N 时,很不幸,在执行到这段代码时app就crash了,crash的原因便是 `FileUriExposedException`。OK,把代码修改下,使用 `ContentProvider` 方式传递uri,这样在Android N上便可以正常运行了。

```
Intent intent = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
ContentValues contentValues = new ContentValues(1);
contentValues.put(MediaStore.Images.Media.DATA, sdcardsTempFile.getAbsolutePath());
Uri uri = context.getContentResolver().insert(MediaStore.Images.Media.EXTERNAL_CONTENT_URI,
```

- `DownloadManager` (<http://developer.android.com/reference/android/app/DownloadManager.html>)不能再使用文件名共享私有存储文件了。老的应用程序可能会因为访问 `COLUMN_LOCAL_FILENAME` ([http://developer.android.com/reference/android/app/DownloadManager.html#COLUMN\\_LOCAL\\_FILENAME](http://developer.android.com/reference/android/app/DownloadManager.html#COLUMN_LOCAL_FILENAME))而终止运行。`targetSdk`为N或者以后版本的app会在访问 `COLUMN_LOCAL_FILENAME` ([http://developer.android.com/reference/android/app/DownloadManager.html#COLUMN\\_LOCAL\\_FILENAME](http://developer.android.com/reference/android/app/DownloadManager.html#COLUMN_LOCAL_FILENAME))时触发 `SecurityException`



(<http://developer.android.com/reference/java/lang/SecurityException.html>)。老的应用程序如果是通过 `setDestinationInExternalFilesDir(Context, String, String)` ([http://developer.android.com/reference/android/app/DownloadManager.Request.html#setDestinationInExternalFilesDir\(android.content.Context,%20java.lang.String,%20java.lang.String\)\)](http://developer.android.com/reference/android/app/DownloadManager.Request.html#setDestinationInExternalFilesDir(android.content.Context,%20java.lang.String,%20java.lang.String))) 或者 `setDestinationInExternalPublicDir(String, String)` ([http://developer.android.com/reference/android/app/DownloadManager.Request.html#setDestinationInExternalPublicDir\(java.lang.String,%20java.lang.String\)\)](http://developer.android.com/reference/android/app/DownloadManager.Request.html#setDestinationInExternalPublicDir(java.lang.String,%20java.lang.String))))将下载路径设置为公共存储区的话仍然可以访问 `COLUMN_LOCAL_FILENAME` ,但官方强烈不建议使用该方式了。更好的方式是通过 `openFileDescriptor(Uri, String)` ([http://developer.android.com/reference/android/content/ContentResolver.html#openFileDescriptor\(android.net.Uri,%20java.lang.String\)\)](http://developer.android.com/reference/android/content/ContentResolver.html#openFileDescriptor(android.net.Uri,%20java.lang.String))))访问 `DownloadManager` 暴露出的文件。

## 辅助功能改进

Android N针对低视力或弱视用户做了一些改进,虽然这些改变不需要开发者修改app的代码,但开发者仍然需要查看这些特性并进行测试,从而评估这些更改对用户体验带来的潜在影响。

## 屏幕缩放

Android N允许用户缩放屏幕上的所有元素,从而提高对视力不佳的用户的设备可用性,但是用户不能缩放到宽度小于`sw320dp`,这是Nexus 4手机的宽度,也是常见的中型手机尺寸。

## NDK Apps Linking to Platform Libraries

这里实在是不知道如何翻译了,就直接用英文标题吧

Android N对native方法的使用作出了严格的限制,namespace的改变将阻止使用非公有(non-public)的c/c++ API,因此你的native代码就只能使用android平台提供的共有API,在Android N的正式版中使用non-public的API会导致app crash。

为了警告你使用了non-public的API,运行在Android N设备上的app在调用non-public的API时,logcat会打出error信息,这条错误信息同时也会显示在屏幕上,以帮助提高对这种情况的认知。开发者应该认真检查native代码以确保移除了non-public API的调用,然后在Android N的设备或模拟器上进行彻底的测试。

如果你的app依赖了平台库,请查阅典型修复的NDK文档,然后用相应的公共API替换掉私有API。你也有可能链接到了平台库但是没有意识到这一点,尤其是你的app使用到的库一部分是平台库,但却不是NDK的一部分。

**注意:** 一些第三方库可能使用了non-public API. 如果你的app用到了这些第三方库,那么在Android N正式设备运行时你的app将有可能crash.

Android技术沉淀 (/nb/3576659)

举报文章 © 著作权归作者所有



逍遥wqy (/u/c71b6a60e9f8)

写了 9025 字, 被 25 人关注, 获得了 33 个喜欢  
(/u/c71b6a60e9f8)

+ 关注

喜欢 (/sign\_in?utm\_source=desktop&utm\_medium=not-signed-in-like-button) | 7





更多分享



(<http://cwb.assets.jianshu.io/notes/images/3602641>)

被以下专题收入，发现更多相似内容

-  Android知识 (/c/3fde3b545a35?utm\_source=desktop&utm\_medium=notes-included-collection)
-  Android... (/c/285f6aa34d80?utm\_source=desktop&utm\_medium=notes-included-collection)

