Android 6.0 Changes

Along with new features and capabilities, Android 6.0 (API level 23) includes a variety of system changes and API behavior changes. This document highlights some of the key changes that you should understand and account for in your apps.

If you have previously published an app for Android, be aware that these changes in the platform affect your app.

Android 6.0增加了一些新的特性和功能，同时修改了部分API的行为方式，也修改了系统的部分地方。本文强调了部分关键修改点，对于这些更改，需要理解并且在APP中考虑到。

另外，请注意平台的这些更改会影响到之前发布过了Android app。

Runtime Permissions

运行时权限认证

This release introduces a new permissions model, where users can now directly manage app permissions at runtime. This model gives users improved visibility and control over permissions, while streamlining the installation and auto-update processes for app developers. Users can grant or revoke permissions individually for installed apps.

On your apps that target Android 6.0 (API level 23) or higher, make sure to check for and request permissions at runtime. To determine if your app has been granted a permission, call the new[checkSelfPermission()](https://developer.android.com/reference/android/content/Context.html#checkSelfPermission(java.lang.String)) method. To request a permission, call the new [requestPermissions()](https://developer.android.com/reference/android/app/Activity.html" \l "requestPermissions(java.lang.String[], int)) method. Even if your app is not targeting Android 6.0 (API level 23), you should test your app under the new permissions model.

For details on supporting the new permissions model in your app, see [Working with System Permissions](https://developer.android.com/training/permissions/index.html). For tips on how to assess the impact on your app, see [Permissions Best Practices](https://developer.android.com/training/permissions/best-practices.html#testing)

这个版本引进一个新的权限模型：用户可以在运行时直接管理app权限。这个模式为用户提供一种改进的可见性的和权限控制，并为开发者提供精简安装和自动升级流程。用户可以对已安装的apps授予或撤销个别权限。

Android 6.0(API level 23)或以上，需要在运行的时候检查或者请求权限。查看自己是否获得某个权限，使用新增的[checkSelfPermission()](https://developer.android.com/reference/android/content/Context.html#checkSelfPermission(java.lang.String))方法。申请某个权限，使用新增的[requestPermissions()](https://developer.android.com/reference/android/app/Activity.html#requestPermissions(java.lang.String[], int)) 方法。就算APP的目标版本不是6.0，也需要测试app是否获得某个权限。

关于这个新的权限模式，参考[Working with System Permissions](https://developer.android.com/training/permissions/index.html).

关于如何评估对ａｐｐ的影响，参考[Permissions Best Practices](https://developer.android.com/training/permissions/best-practices.html#testing).

Doze and App Standby

This release introduces new power-saving optimizations for idle devices and apps. These features affect all apps so make sure to test your apps in these new modes.

* **Doze**: If a user unplugs a device and leaves it stationary, with its screen off, for a period of time, the device goes into Doze mode, where it attempts to keep the system in a sleep state. In this mode, devices periodically resume normal operations for brief periods of time so that app syncing can occur and the system can perform any pending operations.
* **App Standby**: App Standby allows the system to determine that an app is idle when the user is not actively using it. The system makes this determination when the user does not touch the app for a certain period of time. If the device is unplugged, the system disables network access and suspends syncs and jobs for the apps it deems idle.

To learn more about these power-saving changes, see [Optimizing for Doze and App Standby](https://developer.android.com/training/monitoring-device-state/doze-standby.html)

这个版本引进了一种新的最优节能方案来管理空闲的设备和ａｐｐ。这些特性影响所有的ａｐｐ，所以需要在新的模式下测试之前的ａｐｐ．

* **Doze**:如果用户在一段时间内拔掉充电插头或静置设备或关屏，设备将进入Doze模式，这个模式下系统将进入睡眠状态。这个模式下，设备会周期性的恢复普通操作一段时间，那样app能够进行同步，系统也能执行其他阻塞的操作。
* **App Standby**:App Standby用于系统决定某个用户没有激活使用的App是否空闲。系统通过一段时间内用户没有点击这个App来确定某个App是否处于空闲状态。如果设备没有插电，系统认为应该闲置，它将关闭网络、阻塞app的同步和任务。

更多的关于节能的更改，参考[Optimizing for Doze and App Standby](https://developer.android.com/training/monitoring-device-state/doze-standby.html).

Apache HTTP Client Removal

Android 6.0 release removes support for the Apache HTTP client. If your app is using this client and targets Android 2.3 (API level 9) or higher, use the[HttpURLConnection](https://developer.android.com/reference/java/net/HttpURLConnection.html) class instead. This API is more efficient because it reduces network use through transparent compression and response caching, and minimizes power consumption. To continue using the Apache HTTP APIs, you must first declare the following compile-time dependency in yourbuild.gradle file:

android {  
    useLibrary 'org.apache.http.legacy'  
}

6.0版本移除了Apache HTTP client的支持。如果在使用了这个库，或者目标版本为2.3或更高，使用[HttpURLConnection](https://developer.android.com/reference/java/net/HttpURLConnection.html)  替换。这个API更有效，因为它通过使用对上层透明的压缩算法、应答缓存和最小化的电量损耗来减少网络占用。

如果要继续使用Apache HTTP API，需要在build.gradle文件中声明编译的依赖项。

android {  
    useLibrary 'org.apache.http.legacy'  
}

BoringSSL

Android不再使用OpenSLL，改用BoringSSL库。如果在app中使用了Android NDK，不要使用不是NDK API一部分的库，比如libcrypto.so和libssl.so。这些库不是公共API，并且不同的版本或者设备有可能改变或者没有通知的中断使用。此外，对app需要进行安全漏洞检查。用 连接静态加密库的方式 来 替代 另一种本地代码通过JNI调用Java加密API 的方式 作为实践中的更佳选择。

Access to Hardware Identifier

为了给用户提供更为好的数据保护，Android移除了通过可编程的方式来访问设备的硬件本地标示符，比如Wi-Fi和蓝牙API。 [WifiInfo.getMacAddress()](https://developer.android.com/reference/android/net/wifi/WifiInfo.html#getMacAddress())和[BluetoothAdapter.getAddress()](https://developer.android.com/reference/android/bluetooth/BluetoothAdapter.html#getAddress())方法现在返回的是一个常量值：02:00:00:00:00:00.

通过蓝牙和Wi-Fi扫描可以访问附近外部设备的硬件标示符，使用下面方法时必须具有[ACCESS\_FINE\_LOCATION](https://developer.android.com/reference/android/Manifest.permission.html#ACCESS_FINE_LOCATION) 和[ACCESS\_COARSE\_LOCATION](https://developer.android.com/reference/android/Manifest.permission.html#ACCESS_COARSE_LOCATION)权限：

* [WifiManager.getScanResults()](https://developer.android.com/reference/android/net/wifi/WifiManager.html#getScanResults())
* [BluetoothDevice.ACTION\_FOUND](https://developer.android.com/reference/android/bluetooth/BluetoothDevice.html#ACTION_FOUND)
* [BluetoothLeScanner.startScan()](https://developer.android.com/reference/android/bluetooth/le/BluetoothLeScanner.html#startScan(android.bluetooth.le.ScanCallback))

注意：当Android 6.0设备在初始化Wi-Fi或蓝牙扫描时，这个操作对于外部设备来说是可见的，它会生成一个随机的MAC地址。

Notifications

这个版本移除了Notification.setLatestEventInfo()方法。使用[Notification.Builder](https://developer.android.com/reference/android/app/Notification.Builder.html)类作为构建方法。复用[Notification.Builder](https://developer.android.com/reference/android/app/Notification.Builder.html)实例作为周期性通知。调用build()方法来得到更新了的Notification实例。 adb shell dumpsys notification 命令不在打印出通知文本。相应的使用adb shell dumpsys notification –noredact命令来打印通知的文本。

AudioManager Changes

通过[AudioManager](https://developer.android.com/reference/android/media/AudioManager.html) 直接设置音量或者静音不再有效。[setStreamSolo()](https://developer.android.com/reference/android/media/AudioManager.html#setStreamSolo(int, boolean)) 方法过时，而是使用 [requestAudioFocus()](https://developer.android.com/reference/android/media/AudioManager.html#requestAudioFocus(android.media.AudioManager.OnAudioFocusChangeListener, int, int))方法替换。类似的，[setStreamMute()](https://developer.android.com/reference/android/media/AudioManager.html#setStreamMute(int, boolean)) 用[adjustStreamVolume()](https://developer.android.com/reference/android/media/AudioManager.html#adjustStreamVolume(int, int, int))替换，参数为[ADJUST\_MUTE](https://developer.android.com/reference/android/media/AudioManager.html#ADJUST_MUTE) 和[ADJUST\_UNMUTE](https://developer.android.com/reference/android/media/AudioManager.html#ADJUST_UNMUTE)。

Text Selection

https://developer.android.com/images/android-6.0/text-selection.gif

当用户在app中选择选择文本时，现在可以在悬浮按钮中显示对选中文本的操作：剪切、拷贝、复制和包含在悬浮按钮中的查找、评论、删除。 这个用户交互的实现和action bar类似，参考单一的视图实现上下文操作模式：[Enabling the contextual action mode for individual views](https://developer.android.com/guide/topics/ui/menus.html#CABforViews)

实现文本选择悬浮工具按钮，需要在app中改变以下几点：

1.在View或者Activity对象中，将startActionMode(Callback)改成startActionMode(Callback, ActionMode.TYPE\_FLOATING)来改变[ActionMode](https://developer.android.com/reference/android/view/ActionMode.html) 。

2. ActionMode.Callback 的实现改成继承[ActionMode.Callback2](https://developer.android.com/reference/android/view/ActionMode.Callback2.html)。

3.覆盖 [onGetContentRect()](https://developer.android.com/reference/android/view/ActionMode.Callback2.html#onGetContentRect(android.view.ActionMode, android.view.View, android.graphics.Rect)) 方法获得界面中内容坐标的[Rect](https://developer.android.com/reference/android/graphics/Rect.html) 对象（比如文本选择矩形框）。

4.如果矩形位置不再可用并且是失效的唯一的变量，调用[invalidateContentRect()](https://developer.android.com/reference/android/view/ActionMode.html#invalidateContentRect()) 方法。

如果正在使用[Android Support Library](https://developer.android.com/tools/support-library/index.html) 22.2版本，注意悬浮工具栏不是向后兼容并且默认AppCompat库控制[ActionMode](https://developer.android.com/reference/android/view/ActionMode.html) 对象。这可以防止悬浮按钮工具栏消失。在[AppCompatActivity](https://developer.android.com/reference/android/support/v7/app/AppCompatActivity.html)中通过调用[getDelegate()](https://developer.android.com/reference/android/support/v7/app/AppCompatActivity.html#getDelegate())方法返回[AppCompatDelegate](https://developer.android.com/reference/android/support/v7/app/AppCompatDelegate.html) 对象,然后调用[setHandleNativeActionModesEnabled()](https://developer.android.com/reference/android/support/v7/app/AppCompatDelegate.html#setHandleNativeActionModesEnabled(boolean))方法并传入参数false来开启[ActionMode](https://developer.android.com/reference/android/view/ActionMode.html)的支持。这个调用 返回了framework中ActionMode对象的的控制。Android 6.0设备中，framework支持[ActionBar](https://developer.android.com/reference/android/support/v7/app/ActionBar.html) 或者悬浮工具栏模式，但是Android 5.1或更低中，只有ActionBar的支持。

Browser Bookmark Changes

这个版本移除了对全局书签的支持。方法android.provider.Browser.getAllBookmarks() 和android.provider.Browser.saveBookmark()已经被移除。READ\_HISTORY\_BOOKMARKS 和WRITE\_HISTORY\_BOOKMARKS权限也被移除。Android 6.0以后，不能通过系统的provider或申请书签权限，而应该让app自己保存书签。

Android Keystore Changes

[Android Keystore provider](https://developer.android.com/training/articles/keystore.html)不再支持DSA加密算法，ECDSA继续支持。当安全锁屏禁用或者重置（比如被用户或者设备管理员）的时候，不再需要被用来加密的密钥不再删除。

Wi-Fi and Networking Changes

Wi-Fi和网络API的变化如下：

* [WifiConfiguration](https://developer.android.com/reference/android/net/wifi/WifiConfiguration.html) 的状态只有在创建这个对象之后才能被修改。
* 之前，通过[enableNetwork()](https://developer.android.com/reference/android/net/wifi/WifiManager.html#enableNetwork(int, boolean)) 方法并设置disableAllOthers=true ，设备将从别的网络（比如移动网络）断开，然后连接一个特定的Wi-Fi。这个版本，设备不在断开别的网络连接。如果app的目标版本是20或更低，设备将连接选定的Wi-Fi网络。如果app的版本是21或更高，使用多种网络API（比如[openConnection()](https://developer.android.com/reference/android/net/Network.html#openConnection(java.net.URL)), [bindSocket()](https://developer.android.com/reference/android/net/Network.html#bindSocket(java.net.Socket)), 和 新的 [bindProcessToNetwork()](https://developer.android.com/reference/android/net/ConnectivityManager.html#bindProcessToNetwork(android.net.Network))方法 ）来确定网络流量走的是选定的网络。

Camera Service Changes

在这个版本，摄像头服务中的访问共享资源模式由“先来先服务”的访问模式改为“高优先级先服务“的模式。变更点如下：

* 访问摄像头子资源系统，包括打开、管理摄像头设备，将以优先级顺序对子程序排序。用户可见、前台activities获得更高的优先级，得到更高可靠性的资源请求和设备使用服务。
* 正在使用camera服务的低优先级的app有可能被高优先级的app剥夺使用权。在过时的[Camera](https://developer.android.com/reference/android/hardware/Camera.html) API中，[onError()](https://developer.android.com/reference/android/hardware/Camera.ErrorCallback.html#onError(int, android.hardware.Camera)) 方法将被剥夺的app调用，返回被剥夺的结果。而在[Camera2](https://developer.android.com/reference/android/hardware/camera2/package-summary.html) API中，[onDisconnected()](https://developer.android.com/reference/android/hardware/camera2/CameraDevice.StateCallback.html#onDisconnected(android.hardware.camera2.CameraDevice)) 方法将被调用。
* 有的摄像头硬件可以同时为多个程序提供独立服务，但是，多个进程同时使用必然会导致性能的显著下降。摄像头服务目前是会检测并不允许多进程访问摄像头服务的。这个更改可能导致低优先级的进程永远不能获得使用权，就算没有其他app视图访问同一相机设备。
* 切换当前用户将导致上一个用户的app中使用摄像头的客户进程被中止。用户配置文件中声明了当前用户访问摄像头的权限。在实践中，这可以看做是一个来宾账户。当用户进行账户切换时，正在使用摄像头的进程也会中止。

Runtime

ART运行时现在可以正确的实现[newInstance()](https://developer.android.com/reference/java/lang/reflect/Constructor.html#newInstance(java.lang.Object...)) 方法的访问规则这个更改修正了Dalvik在上一个版本中不能正确的检查访问权限规则的问题。如果在app中使用[newInstance()](https://developer.android.com/reference/java/lang/reflect/Constructor.html#newInstance(java.lang.Object...)) 方法，或者想重写访问检查，调用[setAccessible()](https://developer.android.com/reference/java/lang/reflect/AccessibleObject.html#setAccessible(boolean))方法，并且传入参数true。如果app中使用了[v7 appcompat library](https://developer.android.com/tools/support-library/features.html#v7-appcompat) 或者[v7 recyclerview library](https://developer.android.com/tools/support-library/features.html" \l "v7-recyclerview)，必须升级这些库到最新版本。此外，确认XML中声明的其他自定义类已经升级，那样类的构造方法才能被访问。

这个版本升级了动态链接器的行为方式。动态连接器可以分辨一个库的别名和它的路径（[public bug 6670](https://code.google.com/p/android/issues/detail?id=6670)），也实现了通过别名查找。以前的app如果有糟糕的 DT\_NEEDED入口（通常是构建机器文件系统的绝对路径）将有可能在加载的时候失败。

dlopen(3) 的RTLD\_LOCAL 标志现在已经被正确的实现。调用dlopen(3) 的时候 如果不显式的声明，默认的是RTLD\_LOCAL（除非显式的使用RTLD\_GLOBAL）。使用 RTLD\_LOCAL标志，调用 dlopen(3)标志位将不可用于之后加载的库（与 DT\_NEEDED 入口相反）。

在之前的Android版本，如果app需要系统去加载代码重定位的共享库，系统虽然会显示警告但是依然允许去加载库。

这个版本开始，系统拒绝这个库，如果你的app目标版本为23或更高的话。为了帮助检查一个类是否加载失败，你的app必须输出dlopen(3)的失败日志，并且包含dlopen(3)返回的问题描述文本。更多的关于代码重定位，参考 [guide](https://wiki.gentoo.org/wiki/Hardened/Textrels_Guide).

APK Validation

这个平台现在将进行严格的APK验证。一个APK将被认为是损坏的，如果一个在manifest中被声明的文件，在APK中不存在。APK应该被重新签名，如果有任何的内容被移除。

USB Connection

通过USB的设备连接将默认的是仅充电模式。访问设备以及它的内容，用户必须显式的为这些操作授予权限。如果你的app支持通过USB口的用户交互，考虑一下这个交互必须显式的启用

Android for Work Changes

这个版本的企业Android的行为方式包括了以下更改：

* 个人情境中的企业通讯录。当用户查看通话历史时，Google Dialer Call Log 现在是可以显示企业通讯录。设置[setCrossProfileCallerIdDisabled()](https://developer.android.com/reference/android/app/admin/DevicePolicyManager.html#setCrossProfileCallerIdDisabled(android.content.ComponentName, boolean))为true来在Google Dialer Call Log中隐藏工作配置通讯录。仅当你设置[setBluetoothContactSharingDisabled()](https://developer.android.com/reference/android/app/admin/DevicePolicyManager.html#setBluetoothContactSharingDisabled(android.content.ComponentName, boolean))为false的时候，企业通讯录能和个人通讯录一样通过蓝牙在设备间显示.默认情况下，这个参数被设置为true。
* **Wi-Fi配置移除：通过拥有者的配置文件添加的Wi-Fi配置现在将被移除，如果那个业务配置文件被删除的话。**
* **Wi-**Fi配置锁定：活动的设备拥有者创建的Wi-Fi配置不再能被修改或者删除，如果[WIFI\_DEVICE\_OWNER\_CONFIGS\_LOCKDOWN](https://developer.android.com/reference/android/provider/Settings.Global.html#WIFI_DEVICE_OWNER_CONFIGS_LOCKDOWN) 是非零的话。用户依然可以创建和修改他们拥有的Wi-Fi配置。活动的设备拥有者有编辑和移除Wi-Fi配置的权限，就算这些Wi-Fi配置不是他创建的。
* **通过谷歌账户下载DPC（**device policy controller**）：通过DPC app请求管理的谷歌账户被添加到一个超出管理范围的设备中，这个添加的账户现在将显示用户需要安装正确的WPC（？）。这个情况同样适用于通过设置>账户 或者 设备安装初始化向导来添加账户。**
* [DevicePolicyManager](https://developer.android.com/reference/android/app/admin/DevicePolicyManager.html)API的行为方式更改：
  + [setCameraDisabled()](https://developer.android.com/reference/android/app/admin/DevicePolicyManager.html#setCameraDisabled(android.content.ComponentName, boolean)) 的调用将只影响调用这个方法的用户。从管理配置文件中调用这个方法将不会影响主要用户的相机app。
  + 此外，[setKeyguardDisabledFeatures()](https://developer.android.com/reference/android/app/admin/DevicePolicyManager.html" \l "setKeyguardDisabledFeatures(android.content.ComponentName, int)) 方式只对配置的拥有者可用，设备拥有者也一样。
  + 配置拥有者能够设置这些锁屏策略：
    - [KEYGUARD\_DISABLE\_TRUST\_AGENTS](https://developer.android.com/reference/android/app/admin/DevicePolicyManager.html#KEYGUARD_DISABLE_TRUST_AGENTS) and [KEYGUARD\_DISABLE\_FINGERPRINT](https://developer.android.com/reference/android/app/admin/DevicePolicyManager.html#KEYGUARD_DISABLE_FINGERPRINT)，将影响父用户配置的锁屏设置。
    - [KEYGUARD\_DISABLE\_UNREDACTED\_NOTIFICATIONS](https://developer.android.com/reference/android/app/admin/DevicePolicyManager.html#KEYGUARD_DISABLE_UNREDACTED_NOTIFICATIONS)将只影响在管理配置文件名单中的程序。
  + DevicePolicyManager.createAndInitializeUser() and DevicePolicyManager.createUser()已经过时。
  + [setScreenCaptureDisabled()](https://developer.android.com/reference/android/app/admin/DevicePolicyManager.html#setScreenCaptureDisabled(android.content.ComponentName, boolean))方法现在也阻塞辅助结构，当用户给出的app在前台。
  + [EXTRA\_PROVISIONING\_DEVICE\_ADMIN\_PACKAGE\_CHECKSUM](https://developer.android.com/reference/android/app/admin/DevicePolicyManager.html#EXTRA_PROVISIONING_DEVICE_ADMIN_PACKAGE_CHECKSUM)现在默认的是SHA-256。SHA-1依然向后兼容，未来将会移除。[EXTRA\_PROVISIONING\_DEVICE\_ADMIN\_SIGNATURE\_CHECKSUM](https://developer.android.com/reference/android/app/admin/DevicePolicyManager.html#EXTRA_PROVISIONING_DEVICE_ADMIN_SIGNATURE_CHECKSUM)现在只支持SHA-256
  + 设备初始化API在Android 6.0中移除
  + EXTRA\_PROVISIONING\_RESET\_PROTECTION\_PARAMETERS移除，所以NFC碰撞检测不能以编程的方式解锁 一个工厂重置保护设备。
  + 额外的，可以通过[EXTRA\_PROVISIONING\_ADMIN\_EXTRAS\_BUNDLE](https://developer.android.com/reference/android/app/admin/DevicePolicyManager.html#EXTRA_PROVISIONING_ADMIN_EXTRAS_BUNDLE)在NFC就绪的设备的app间进行数据交换。
  + Android 企业版API对于M运行时权限进行了最优化，包括企业级配置文件、辅助层或者其他。 新的[DevicePolicyManager](https://developer.android.com/reference/android/app/admin/DevicePolicyManager.html)API不会影响M之前的app.
  + 当通过包含[ACTION\_PROVISION\_MANAGED\_PROFILE](https://developer.android.com/reference/android/app/admin/DevicePolicyManager.html#ACTION_PROVISION_MANAGED_PROFILE) 或者[ACTION\_PROVISION\_MANAGED\_DEVICE](https://developer.android.com/reference/android/app/admin/DevicePolicyManager.html" \l "ACTION_PROVISION_MANAGED_DEVICE)标记的Intent来取消已经开始的安装流程中的同步部分 ，系统现在将返回[RESULT\_CANCELED](https://developer.android.com/reference/android/app/Activity.html#RESULT_CANCELED)结果码。
* **Changes to other APIs**:
  + 数据利用率：android.app.usage.NetworkUsageStats类重命名为[NetworkStats](https://developer.android.com/reference/android/app/usage/NetworkStats.html)。
* **Changes to global settings**:
  + 全局设置不再通过[setGlobalSettings()](https://developer.android.com/reference/android/app/admin/DevicePolicyManager.html#setGlobalSetting(android.content.ComponentName, java.lang.String, java.lang.String))来设置：
    - BLUETOOTH\_ON
    - DEVELOPMENT\_SETTINGS\_ENABLED
    - MODE\_RINGER
    - NETWORK\_PREFERENCE
    - WIFI\_ON
  + These global settings can now be set via [setGlobalSettings()](https://developer.android.com/reference/android/app/admin/DevicePolicyManager.html#setGlobalSetting(android.content.ComponentName, java.lang.String, java.lang.String)):
    - [WIFI\_DEVICE\_OWNER\_CONFIGS\_LOCKDOWN](https://developer.android.com/reference/android/provider/Settings.Global.html#WIFI_DEVICE_OWNER_CONFIGS_LOCKDOWN)