licaomeng的专栏

Focus on Android, Front-end

个人资料

licaomengRICE

访问： 29618次
积分： 527
等级： BLOG 3
排名： 千里之外

原创： 17篇 转载： 0篇
译文： 1篇 评论： 47条

深入Android 'M' Doze

标签： android doze idle Lollipop android-M

2015-09-25 17:59 2922人阅读 评论(0) 收藏 举报

分类： Android（8）

目录(?) [+]

原文链接：https://newcircle.com/s/post/1739/2015/06/12/diving-into-android-m-doze

> **注意：**这篇博文的所有描述都是基于**Android**‘M’开发者预览的第一版本。因为新的版本（包括码代码）已经发布，一些东西发生了改变，这篇博文的内容也可能会随之发生改变。

什么是Doze？

我第一次看到”doze”被用在Android上，其实是它作为一个display state在搭载了KitKat(Android 4.4, API 20)的穿戴设备上，之后我在搭载了Lollipop(Android 5.0)的设备上又看到了它。Doze是当设备暂时呈现出静态（不交互的）内容

（想象一下Nexus 6，响应重要手势时出现的时钟，实际上并没有更新）时所处的一种全新的、低耗能的状态。在新的Android‘M’预览中，Doze Mode的含义发生了轻微的变化，现在是指“强制空闲”状态，这时只有很少的后台处理被允许。

> **评论：**“Doze”对于这个新特性而言其实是一个糟糕的名字。因为这个名词在已经在framework层中存在了，是由**DreamManager**掌管，而与Doze Mode没有任何关系。

你好，DeviceIdleController

DeviceIdleController是Doze模式的主要驱动。接下来，我将使用***device idle mode***而不是***doze mode***来描述“Doze”，因为它更符合代码的实际情况。如果你已经阅读了官方文档，你可能已经注意到下面的命令，开发者可以通过这些命令得知当下设备的应用行为：

```
$ adb shell dumpsys battery unplug
$ adb shell dumpsys deviceidle step
```

对于上面的命令你可能并不熟悉，**dumpsys**是用来与系统服务交互的（查看它们的状态），**deviceidle**是我有看到过的，它是一个新的系统服务。

```
$ adb shell service list | grep deviceidle
59  deviceidle: [android.os.IDeviceIdleController]
```

这个新的服务常驻以监听下面的系统事件，这些事件会触发系统是否进入idle mode：

1. 亮屏/暗屏
2. 充电状态
3. 重要的手势检测

DeviceIdleController维持着设备包含的五种状态：

评论排行

推荐文章

1. **ACTIVE** – 设备在使用中，或者连接着电源。
2. **INACTIVE** – 设备已经从ACTIVE状态中出来一段时间了（使用者关闭了屏幕或者拔掉了电源）
3. **IDLE_PENDING** – 请留意，我们将进入idle mode.
4. **IDLE** – 设备进入idle mode.
5. **IDLE_MAINTENANCE** – 应用窗口已经打开去做处理.

当设备被唤醒和正在使用中，控制器就处于**ACTIVE**状态，外部的事件（不活跃时间超时，用户关闭屏幕，等等）将会使设备状态进入到**INACTIVE**. 那时，**DeviceIdleController**将会通过**AlarmManager**来设置他自己的alarm来驱动进程：

1. 一个alarm会被设置在一个预设的时刻（这个时间在M的预览中是30分钟）。
2. 当这个alarm生效后，**DeviceIdleController** 会进入到**IDLE_PENDING**然后再次设置同样的alarm。
3. 当触发下一个alarm后，控制器会进入到**IDLE** 状态，进入到这个状态后，应用特性会被完全限制。
4. 再向前推进这个服务会在**IDLE** 和**IDLE_MAINTENANCE**两个状态之间周期性的跳转，后者在服务完全被禁前，等待的应用事件被触发。

正如上面提到的，开发者能够使用下面的命令，手动地改变设备的这些状态：

```
$ adb shell dumpsys deviceidle step
```

我们可以使用‘-h’看到所有的**deviceidle**的所有选项：

```
$ adb shell dumpsys deviceidle -h
Device idle controller (deviceidle) dump options:
 [-h] [CMD]
 -h: print this help text.
Commands:
 step
  Immediately step to next state, without waiting for alarm.
 disable
  Completely disable device idle mode.
 enable
```

* 自定义控件三部曲视图篇（三）——瀑布流容器WaterFallLayout实现

* 面向服务的体系架构（SOA）—架构篇

```
11    Re-enable device idle mode after it had previously been disabled.
12  whitelist
13    Add (prefix with +) or remove (prefix with -) packages.
```

这些服务的公共API（由**IDeviceIdleController** 接口展现）持有全部方法访问白名单。应用（系统应用或其它第三方应用）任何情况下都不能驱动控制器状态。

你在这个名单中吗？

正如你在上面的帮助菜单中看到的一样，**DeviceIdleController** 维护着一个应用白名单，不需要额外的参数，通过dump服务的状态，我们能够看到现在的这个列表：

```
1  $ adb shell dumpsys deviceidle
2    Whitelist system apps:
3      com.android.providers.downloads
4      com.android.vending
5      com.google.android.gms
6    Whitelist app uids:
7      UID=10012: true
8      UID=10016: true
9      UID=10026: true
10     ...
```

这个名单分为两个部分：系统应用和第三方应用。

系统应用

系统应用会被平台制作者通过配置定义列在白名单中。下面这个是从Nexus 6中得到的一个配置定义例子，它将GMS核心（在GCM中使用），应用商店，以及一个任意的用于电源监控的app白名单化：

/system/etc/sysconfig/google.xml

```
<?xml version="1.0" encoding="utf-8"?>
<!-- These are configurations that must exist on all GMS devices. -->
<config>
    <allow-in-power-save package="com.google.android.gms" />

    <allow-in-power-save package="com.android.vending" />

    <allow-in-power-save package="com.google.android.volta" />
</config>
```

其它的系统服务可以通过**SystemConfig**的实例访问到这些值。**DeviceIdleController**使用**SystemConfig.getAllowInPowerSave()**将这些系统定义的元素放到白名单中。

> **注意：**当设备处于“省电模式”时，同样也是这个配置文件决定哪个系统应用可以在后台开启服务。

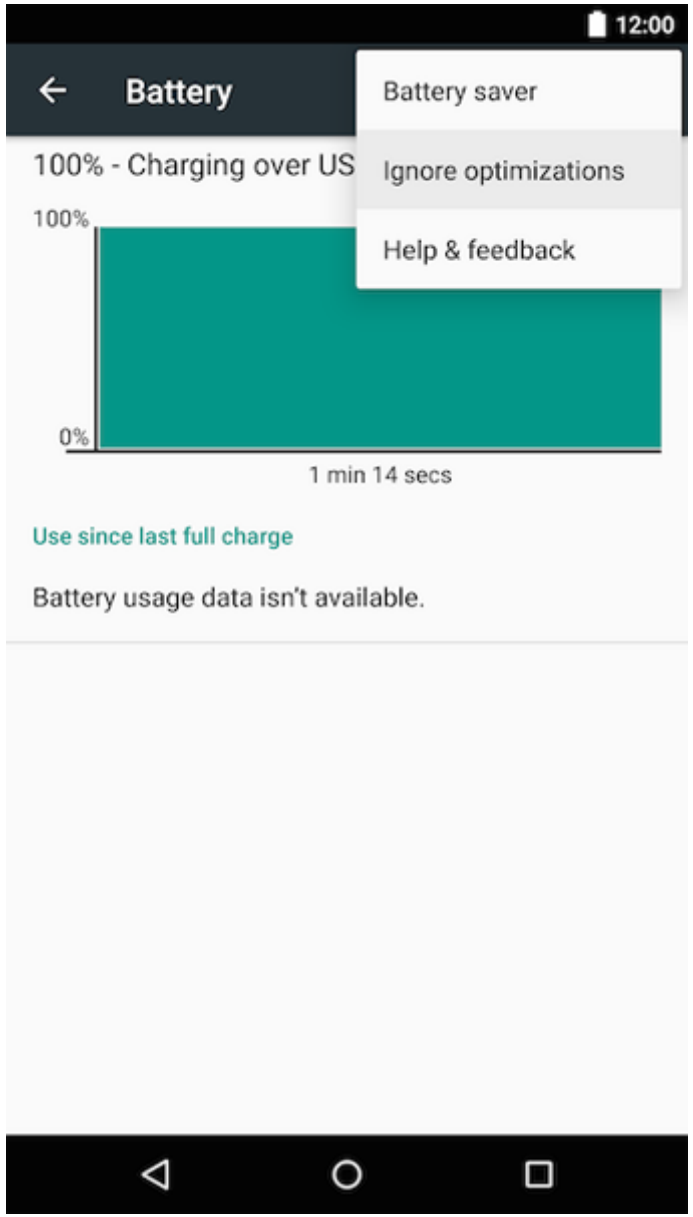
第三方应用

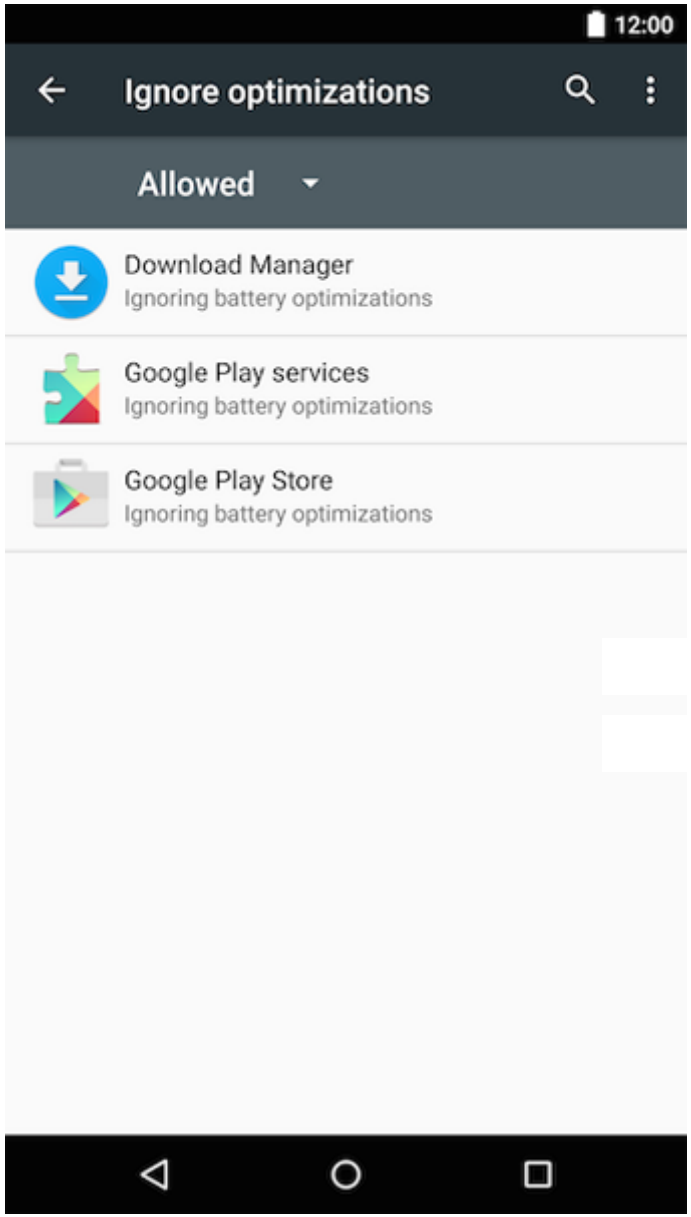
白名单中剩下的部分是用户定义的，这些项可以通过两种方式被增加和删除。

第一种方式，开发者可以通过**dumpsys**接口使用**白名单**命令：

```
$ adb shell dumpsys deviceidle whitelist +com.example.myapplication
$ adb shell dumpsys deviceidle
  Whitelist system apps:
    com.android.providers.downloads
    com.android.vending
    com.google.android.gms
  Whitelist user apps:
    com.example.myapplication
  Whitelist app uids:
    UID=10012: true
```

第二种方式，用户可以通过设置（**Settings -> Battery -> Ignore optimizations**）来修改白名单。





这个用户设置存在是因为白名单也被Android‘M’新的App Standby 特征使用。白名单只是当评估设备的空闲行为时被部分使用。

设备处于Idle Mode时的系统服务行为

既然我们已经知道了一些基础了，让我们来**测试**一下当设备空闲时剩下的系统服务的反应，并且看下白名单是否有实际的用处。下面是**DeviceIdleController**所能涉及到的主要服务的列表：

NetworkPolicyManagerService

Lollipop引入了“省电模式”，它只允许白名单应用才能在后台运行。Idle mode有着相同的逻辑，只对白名单中的应用授予访问网络的权限。这个服务对于“省电模式”和idle mode是等同的。

JobSchedulerService

所有应用（没有例外）中，现在正在执行的作业会被取消。Idle mode的时候没有等待中的作业被开启。

SyncManager

所有应用（没有例外）中的活跃的同步会被取消。

PowerManagerService

应用白名单意味着唤醒锁（wake locks）是有效的。没有在这个名单中的应用及时禁用的唤醒锁（wake locks）

AlarmManagerService

DeviceIdleController是使用一个特殊的私有方法(AlarmManager.setIdleUntil())来注册下一个唤醒alarm。当AlarmManagerService 看到它时，所有的标准应用alarm都强制进入到一个等待状态直到直到下一个DeviceIdleControlleralarm触发。以这种方式，应用alarm被批处理并且在IDLE_MAINTENANCE时期被控制器驱动。有三个标志允许alarm退出。

- **FLAG_ALLOW_WHILE_IDLE** – 任何应用可以使用new setAndAllowWhileIdle() 对其进行设置。
- **FLAG_WAKE_FROM_IDLE** – 任何应用可以使用**setAlarmClock()**来进行设置。

- **FLAG_IDLE_UNTIL** – 为被**DeviceIdleController**使用的alarm保留，以改变机器的状态。
  UID<10000的进程（例如系统服务）他们设置的每个alarm会自动被授予**FLAG_ALLOW_WHILE_IDLE**。

最后总结

许多开发者询问，文档中提到应用在空闲时期，如果接收到更高优先级的使用GCM推送的消息，应用会被授予“短暂的网络访问”。在我看来，在framework层的实现中没有任何与GCM的关联（谢天谢地，如果是那样就太愚蠢了，不是吗？）因为GMS是闭源的，我们只能推测在谷歌设备上会不会发生。

事实是这样，任何系统应用能够暂时修改app白名单或者特殊应用的网络策略。这会在不修改放下设备的空闲状态的情况下，为每个应用基础的网络访问打破限制。在当前的预览中，是否仍然存在漏洞于SyncManager，这点仍然不清楚。现存的同步被取消，但是我们不能确定新的会不会被允许。应用也许仍然能够在GCM的触发下执行新的同步
（这是在这种状态下仅有的app允许的网络访问）
好吧，当然，这纯属推测。

这篇博文是Android M Development Tips 的部分。

（由于本人翻译水平有限，因此译文中不免出现错误和不当之处。还请各位看官不吝指正，以提高译文水平，便更多同学受益）

上一篇 Android Touch事件传递机制
下一篇 手机淘宝的客户端架构探索之路

我的同类文章

Android（8）

- Material Design风格的水波涟.. 2016-02-21 阅读 2206
- 手机淘宝的客户端架构探索之.. 2015-10-19 阅读 3757
- Android中的多线程之AsyncT... 2015-09-13 阅读 355
- Android中java反射（Reflecti... 2015-08-08 阅读 2248
- Android开源项目解析：PullT... 2015-12-17 阅读 365
- Android Touch事件传递机制 2015-09-13 阅读 1668
- Android中的多线程之handler 2015-08-16 阅读 368
- Android中的Context----既熟... 2015-07-29 阅读 337

参考知识库

Android知识库

33156 关注 | 2675 收录

软件测试知识库

4413 关注 | 318 收录

猜你在找

Android开发高级组件与框架——APP上线
Android底层技术：HAL驱动开发
Android底层技术：Linux驱动框架与开发
Android驱动深度开发视频教程
【Android APP开发】Android高级商业布局快速实现
安装Android Studio报failed to find java version for
一起学android之自定义控件一起制作自定义标签39
Android39_Volley网络通信框架
Android入门39第一章 使用Debug方式调试程序
android_39_跳转至第2个Activity

查看评论

暂无评论

您还没有登录,请[登录]或[注册]

* 以上用户言论只代表其个人观点，不代表CSDN网站的观点或立场

核心技术类目

全部主题 Hadoop AWS 移动游戏 Java Android iOS Swift 智能硬件 Docker OpenStack
VPN Spark ERP IE10 Eclipse CRM JavaScript 数据库 Ubuntu NFC WAP jQuer
BI HTML5 Spring Apache .NET API HTML SDK IIS Fedora XML LBS Unity
Splashtop UML components Windows Mobile Rails QEMU KDE Cassandra CloudStack
coremail OPhone CouchBase 云计算 iOS6 Rackspace Web App SpringSide Maemo
Compuware 大数据 aptech Perl Tornado Ruby Hibernate ThinkPHP HBase Pure Solr
Angular Cloud Foundry Redis Scala Django Bootstrap