Android PowerManager 简述

Last edited by **caoquanli** 1 month ago

Android Powermanager

- Designed for mobile device
- Goal is to prolong battery life
- Build on top of Linux Power Management
  - Not directly suitable for mobile device

- Designed for devices which have a 'default-off' behavior

小弟不才翻译了一下

- 为mobile设备设计
- 目标是延长电池使用时间
- 基于Linux电源管理
  - Linux电源管理不直接支持mobile

- 为“默认关闭”的设备设计

PowerManager

PowerManager中开放给应用一系列接口，应用可以调用PM的接口，如

- 申请wakelock
- 唤醒系统
- 进入睡眠

等操作。

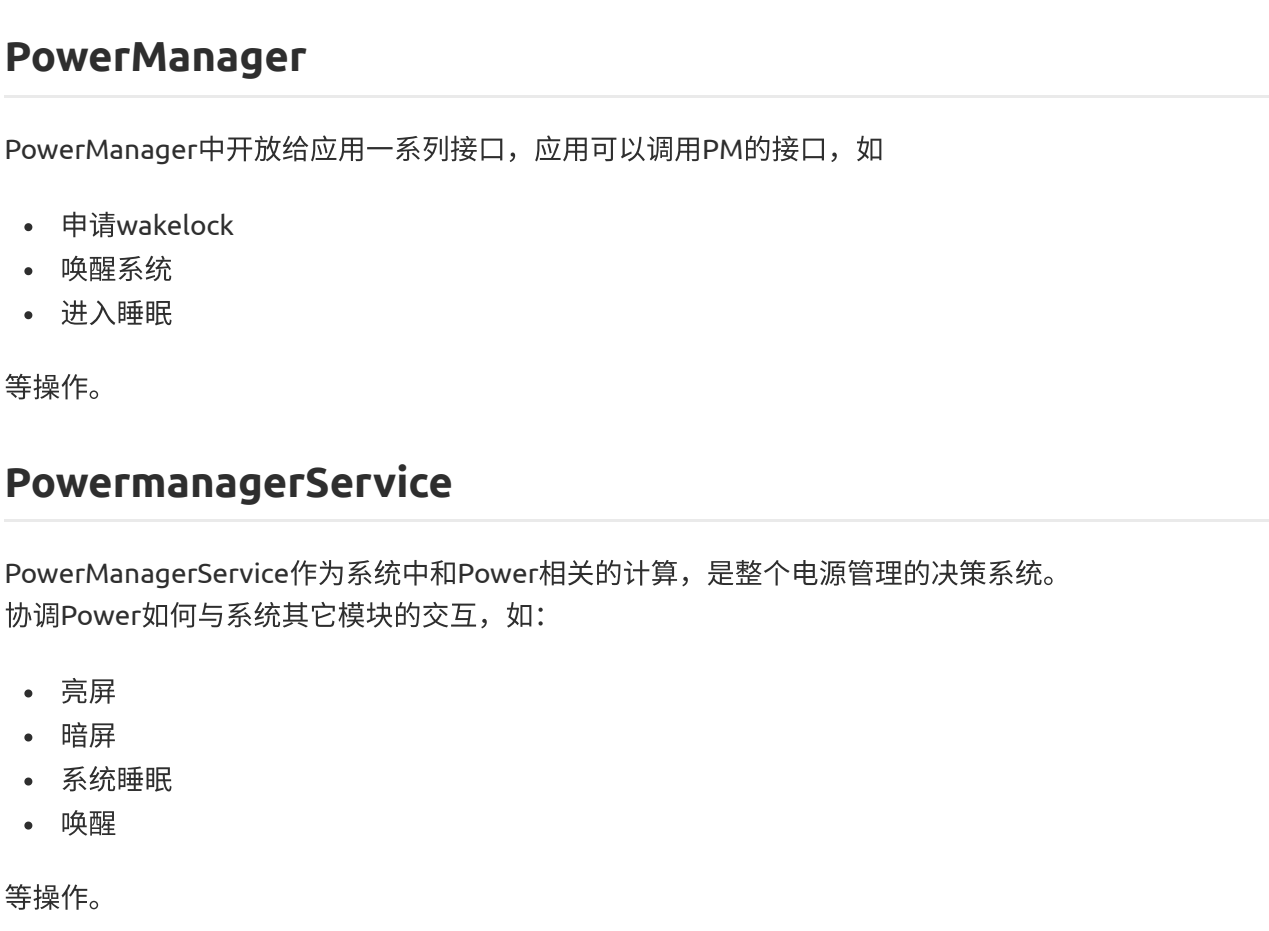
PowermanagerService

PowerManagerService作为系统中和Power相关的计算，是整个电源管理的决策系统。
协调Power如何与系统其它模块的交互，如：

- 亮屏
- 暗屏
- 系统睡眠
- 唤醒

等操作。

Android PowerManger design



- 基于Linux,在kernel层的改变
- 在power state中增加了'on'状态
- 在FW中增加了Early Suspend的概念(已移除)
- 增加了 Partial WakeLock的概念
- Apps 和 Services必须请求 wakelocks 来保证 CPU on, 否则Android将会shutdown CPU
- 使用wakelocks和超时原理切换系统电源状态->降低电量消耗

WakeLock

在介绍powermanagerservice前请先看下wakelock，这是pwms的核心机制。 wake_lock简述

PowerManagerService

PWMS由SystemServer通过反射的方式启动.

首先，在SystemServer的main()方法中，调用了自身的run()方法，并在run()方法中启动三类服务:引导服务、核心服务和其他服务.
引导服务中启动的是一些依赖性比较强的服务，其中就包括了PMS.

PWMS启动流程

1. construction方法

- new thread
- 实例化Handler
- 创建WakelLocks & Display的SuspendBlocker的实例
- 申请Display的SuspendBlocker锁
- 初始化native方法

2. onStart方法

- 将Power服务进行Binder注册(publishBinderService)和本地注册(publishLocalService)
- 设置watchdog监听

3. (多次调用)onBootPhase方法

- mDirty(用于表示电源状态某一部分发生改变)
- 调用updatePowerStateLocked()更新电源状态信息

4. systemReady方法

- 获取本地 & 远程服务
  - DreamManagerService
  - DisplayManagerService
  - WindowManagerService
  - BatteryService
  - BatteryStatsService
  - LightsManager

- 获取屏幕亮度信息
- 获取与其他系统服务间的广播服务
- 获取无线充电相关信息、
- 监听Stetings中值的变化
- initPowerManager注册DisplayPower回调
- 注册用于和其他Sytem交互的广播
- 调用readConfigurationLocked()读取配置文件
- 调用updateSettingsLocked()
  - 屏保是否支持
  - 休眠时是否启用屏保
  - 插入基座时屏保是否激活
  - 设备在一段时间不活动后进入休眠或者屏保状态的超时时间，默认15*1000ms
  - 充电时屏幕一直开启
  - 是否支持剧院模式

    - 屏幕保持常亮
    - 双击唤醒屏幕设置
    - 屏幕亮度,自动调节亮度值(>0.0 <1.0）
    - 重置临时亮度值
    - 亮度调节模式
    - 低电量模式是否可用
    - 更新低电量模式

- 更新mDirty
- 调用updatePowerStateLocked
- 注册SettingsObserver监听

updatePowerStateLocked

updatePowerStateLocked()是整个PWMS的核心方法。用于更新整个电源状态的改变，并重新计算。
PWMS中使用一个int值的mDirty标志位作为判断电源状态是否发生变化的依据。
当电源状态发生改变时，例如亮灭屏、电池状态改变、暗屏等都会调用该方法。

updateWakeLockSummaryLocked

在这个方法中，会对当前所有的WakeLock锁进行统计，过滤所有的wakelock锁状态（wakelock锁机制在后续进行分析），并更新mWakeLockSummary的值以汇总所有活动的唤醒锁的状态。
mWakeLockSummary是一个用来记录所有WakeLock锁状态的标识值，该值在请求Display状时会用到。
当系统处于睡眠状态时，大多数唤醒锁都将被忽略，比如系统在处于唤醒状态(awake)时，会忽略PowerManager.DOZE_WAKE_LOCK类型的唤醒锁，系统在处于睡眠状态(asleep)或者Doze状态时，会忽略PowerManager.SCREEN_BRIGHT类型的锁等等。该方法如下：

```
private void updateWakeLockSummaryLocked(int dirty) {
    if ((dirty & (DIRTY_WAKE_LOCKS | DIRTY_WAKEFULNESS)) != 0) {
    mWakeLockSummary = 0;
    //在waklock集合中遍历wakelock
    final int numWakeLocks = mWakeLocks.size();
    for (int i = 0; i < numWakeLocks; i++) {
        final WakeLock wakeLock = mWakeLocks.get(i);
        switch (wakeLock.mFlags & PowerManager.WAKE_LOCK_LEVEL_MASK) {
            case PowerManager.PARTIAL_WAKE_LOCK:
                if (!wakeLock.mDisabled) {
                    // We only respect this if the wake lock is not disabled.
                    //如果存在PARTIAL_WAKE_LOCK并且该wakelock可用,通过置位进行记录，下同
                    mWakeLockSummary |= WAKE_LOCK_CPU;
                }
                break;
            case PowerManager.FULL_WAKE_LOCK:
                ...
                break;
            ...
    }
}
 //设备不处于DOZE状态时，通过置位操作忽略相关类型wakelock
 //PowerManager.DOZE_WAKE_LOCK和WAKE_LOCK_DRAW锁仅仅在Doze状态下有效
if (mWakefulness != WAKEFULNESS_DOZING) {
    mWakeLockSummary &= ~(WAKE_LOCK_DOZE | WAKE_LOCK_DRAW);
}
...
//根据当前状态推断必要的wakelock
if ((mWakeLockSummary & (WAKE_LOCK_SCREEN_BRIGHT |
        WAKE_LOCK_SCREEN_DIM)) != 0) {
    //处于awake状态，WAKE_LOCK_STAY_AWAKE只用于awake状态时
    if (mWakefulness == WAKEFULNESS_AWAKE) {
        mWakeLockSummary |= WAKE_LOCK_CPU |
        WAKE_LOCK_STAY_AWAKE;
        //处于屏保状态(dream)
```

```
        } else if (mWakefulness == WAKEFULNESS_DREAMING) {
            mWakeLockSummary |= WAKE_LOCK_CPU;
        }
    }
    ...
  }
}
```

当这个方法执行后，将未忽略的wakelocks类型都记录在mWakeLockSummary这个标志位中。
在分析处理不灭屏相关Bug时，通过该值可确定当前系统持有哪些类型的锁。

updateUserActivitySummaryLocked

用来更新用户活动时间，当设备和用户有交互时，都会根据当前时间和休眠时长、Dim时长、所处状态而计算下次休眠的时间，从而完成用户活动超时时的操作。如由亮屏进入Dim的时长、Dim到灭屏的时长、亮屏到屏保的时长，就是在这里计算的。
举个例子，假设设备设定休眠时间为15s，Dim时长为3s，按power键唤醒设备后会计算这个相应的时间。

在计算灭屏超时时间时，有两个值比较重要

```
final int sleepTimeout = getSleepTimeoutLocked();
final int screenOffTimeout = getScreenOffTimeoutLocked(sleepTimeout);
```

其中sleepTimeout表示设备在经过一段不活动后完全进入睡眠后屏保的时间，该值可以理解为保持唤醒或屏保的最大值或上限，并且该值要大于screenOffTimeout,默认为-1，表示禁用此项功能。
screenOffTimeout表示设备在经过一段不活动后进入睡眠或屏保的时间，也称为用户活动超时时间，但屏幕到期时不一定关闭。该值可以在设置-休眠中设置。

updateWakefulnessLocked

之所以把updateWakeLockSummaryLocked()、updateUserActivitySummaryLocked()、updateWakefulnessLocked()这三个方法放在for(;;)循环中调用，是因为它们共同决定了设备的状态，前两个方法是汇总状态，后一个方法是根据前两个方法汇总的值而进行判断是否要改变当前的设备唤醒状态,汇总状态会受mWakefulness的影响，因此会进行循环处理。

updateDisplayPowerStateLocked

用于更新设备显示状态，在这个方法中，会计算出最终需要显示的亮度值和其他值，然后将这些值封装到DisplayPowerRequest对象中，向DisplayMangerService请求Display状态，完成屏幕亮度显示等。

updateSuspendBlockerLocked

Suspend锁机制是Android电源管理框架中的一种机制，在前面还提到的wakelock锁也是，不过wakelock锁是上层向framwork层申请，而Suspend锁是framework层中对wakelock锁的表现，也就是说，上层应用申请了wakelock锁后，在PMS中最终都会表现为Suspend锁，通过Suspend锁向Hal层写入节点，Kernal层会读取节点，从而进入唤醒或者休眠。这个方法就是用来申请Suspend锁操作。

shutdown

shutdown链接

详细的分析在上链接

gotosleep

gotosleep链接

详细的分析在上链接