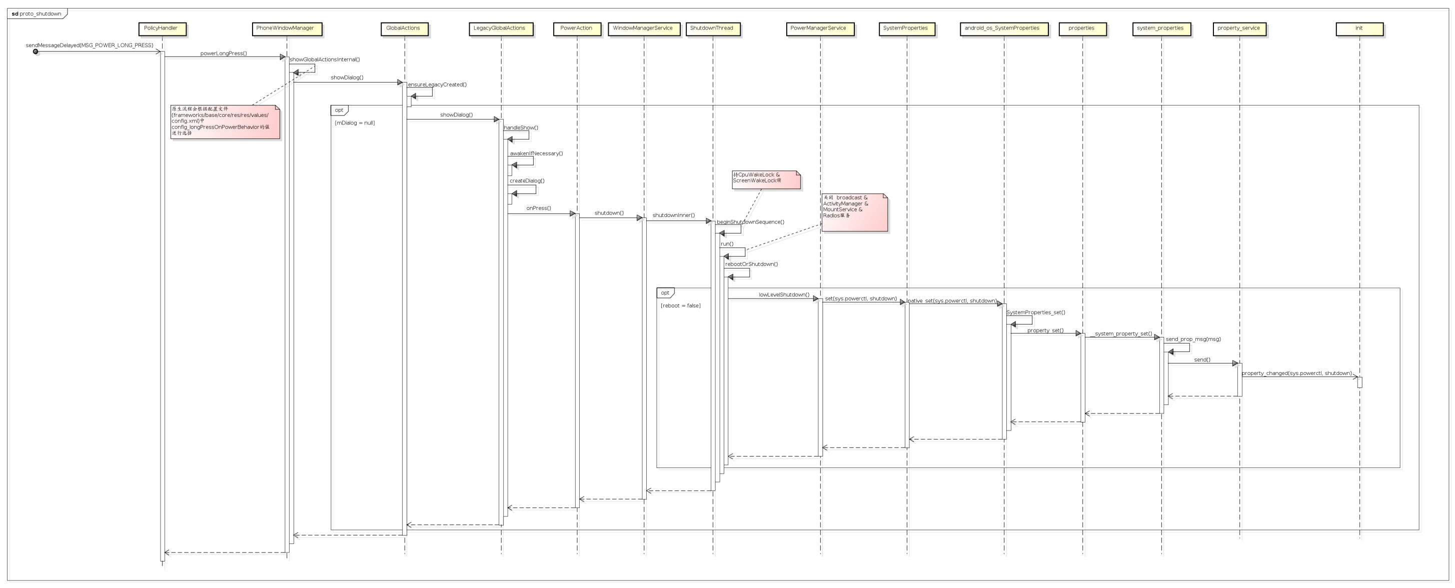
android shutdown流程

Last edited by **caoquanli** 1 month ago



原生shutdown大致流程

1. 接受powerkey的分发
2. 根据powerkey的类型（长按）进入powerlongpress
3. 显示长按关机后的对话框（飞行、关机、重启类似选项）
4. 设置onPress监听用户选择->关机
5. 进入shutdownthread.run

- 5.1 发送关机广播Intent.ACTION_SHUTDOWN
- 5.2 关闭am
- 5.3 关闭pm
- 5.4 关闭radios
- 5.5 关闭StorageManagerService

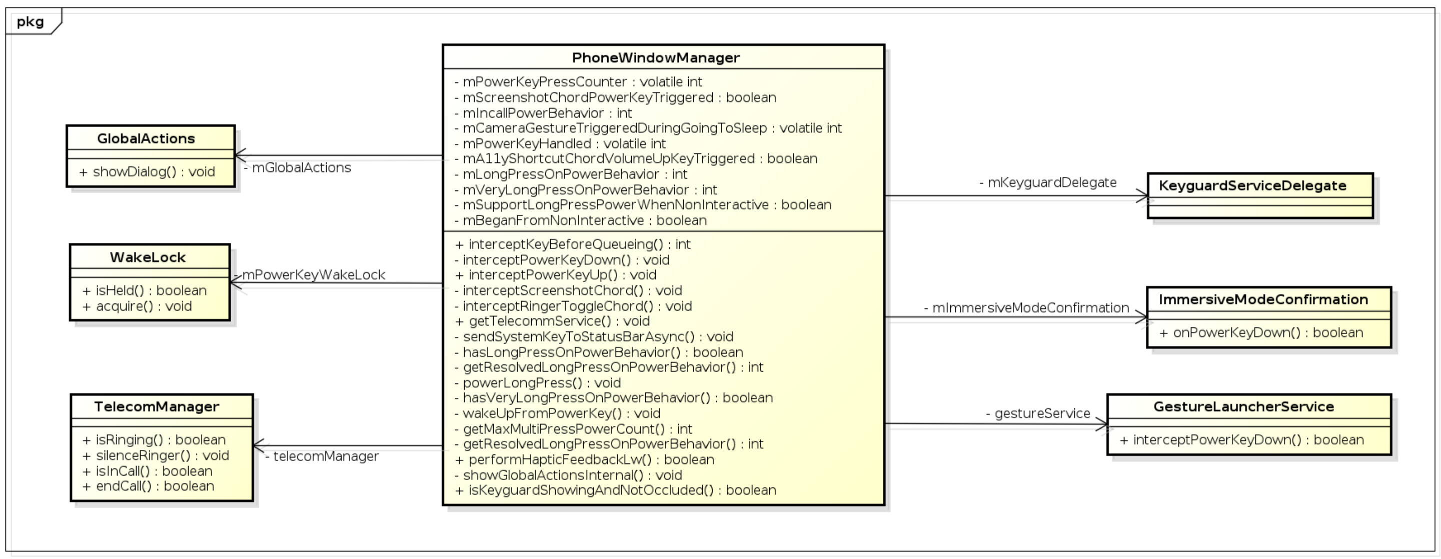
6. 震动
7. 进native后关机

核心类

- interceptPowerKeyDown
- powerLongPress
- shutdownThread.run

STEP1 interceptKeyBeforeDispatching

当长按电源键时，按键消息被分发到PhoneWindowManager的interceptKeyBeforeQueueing函数中处理：


pkg
GlobalActions
+ showDialog() : void
- mGlobalActions
WakeLock
+ isHeld() : boolean
+ acquire() : void
- mPowerKeyWakeLock
TelecomManager
+ isRinging() : boolean
+ silenceRinger() : void
+ isInCall() : boolean
+ endCall() : boolean
- telecomManager
PhoneWindowManager
- mPowerKeyPressCounter : volatile int
- mScreenshotChordPowerKeyTriggered : boolean
- mIncallPowerBehavior : int
- mCameraGestureTriggeredDuringGoingToSleep : volatile int
- mPowerKeyHandled : volatile int
- mA11yShortcutChordVolumeUpKeyTriggered : boolean
- mLongPressOnPowerBehavior : int
- mVeryLongPressOnPowerBehavior : int
- mSupportLongPressPowerWhenNonInteractive : boolean
- mBeganFromNonInteractive : boolean
+ interceptKeyBeforeQueueing() : int
- interceptPowerKeyDown() : void
+ interceptPowerKeyUp() : void
- interceptScreenshotChord() : void
- interceptRingerToggleChord() : void
+ getTelecommService() : void
- sendSystemKeyToStatusBarAsync() : void
- hasLongPressOnPowerBehavior() : boolean
- getResolvedLongPressOnPowerBehavior() : int
- powerLongPress() : void
- hasVeryLongPressOnPowerBehavior() : boolean
- wakeUpFromPowerKey() : void
- getMaxMultiPressPowerCount() : int
- getResolvedLongPressOnPowerBehavior() : int
+ performHapticFeedbackLw() : boolean
- showGlobalActionsInternal() : void
+ isKeyguardShowingAndNotOccluded() : boolean
- mKeyguardDelegate
KeyguardServiceDelegate
- mImmersiveModeConfirmation
ImmersiveModeConfirmation
+ onPowerKeyDown() : boolean
- gestureService
GestureLauncherService
+ interceptPowerKeyDown() : boolean

PhoneWindowManager.java->interceptKeyBeforeDispatching

```
public long interceptKeyBeforeDispatching(WindowState win, KeyEvent event, int policyFlags)
    case KeyEvent.KEYCODE_POWER: {
        result &= ~ACTION_PASS_TO_USER;
        isWakeKey = false; // wake-up will be handled separately
        if (down) {
            interceptPowerKeyDown(event, interactive);
        } else {
            interceptPowerKeyUp(event, interactive, canceled);
        }
        break;
    }
}
```

STEP2 interceptPowerKeyDown

PhoneWindowManager.java->interceptPowerKeyDown

```
private void interceptPowerKeyDown(KeyEvent event, boolean interactive) {
    //..........
    // When interactive, we're already awake.
    // Wait for a long press or for the button to be released to decide what to do.
    if (hasLongPressOnPowerBehavior()) {
    // 如果存在长按事件，就发送MSG_POWER_LONG_PRESS。
        Message msg = mHandler.obtainMessage(MSG_POWER_LONG_PRESS);
        msg.setAsynchronous(true);
        mHandler.sendMessageDelayed(msg, ViewConfiguration.get(mContext).getDeviceGlobalAct
    }
    //........
}
```

STEP3

handler根据msg=MSG_POWER_LONG_PRESS进入powerLongPress方法

PhoneWindowManager$PolicyHandler->handleMessage

```
private class PolicyHandler extends Handler {
    @Override
    public void handleMessage(Message msg) {
        // ......
            case MSG_POWER_LONG_PRESS:
                powerLongPress();    //处理长按事件
                break;
        // ......
        }
    }
}
```

(getResolvedLongPressOnPowerBehavior)如果是工厂测试则直接关机否则返回 mLongPressOnPowerBehavior(`frameworks/base/core/res/res/values/config.xml` 中 config_longPressOnPowerBehavior进行配置)

- LONG_PRESS_POWER_NOTHING = 0 代表不做任动作
- LONG_PRESS_POWER_GLOBAL_ACTIONS = 1 代表是全局的动作,显示关机dialog
- LONG_PRESS_POWER_SHUT_OFF = 2 表示确认后关机(只有关机一个选项)
- LONG_PRESS_POWER_SHUT_OFF_NO_CONFIRM = 3 表示不确认直接关机(默认为1)。

紧接着根据behavior值选择对应操作

PhoneWindowManager.java->powerLongPress

```
private void powerLongPress() {
    final int behavior = getResolvedLongPressOnPowerBehavior();
    switch (behavior) {
    case LONG_PRESS_POWER_NOTHING:
        break;
    case LONG_PRESS_POWER_GLOBAL_ACTIONS:
        mPowerKeyHandled = true;
        if (!performHapticFeedbackLw(null, HapticFeedbackConstants.LONG_PRESS, false)) {
            performAuditoryFeedbackForAccessibilityIfNeed();
        }
        // 我们直接看显示Dialog这一条分支
        showGlobalActionsInternal();
        break;
    case LONG_PRESS_POWER_SHUT_OFF:
    case LONG_PRESS_POWER_SHUT_OFF_NO_CONFIRM:
        mPowerKeyHandled = true;
        performHapticFeedbackLw(null, HapticFeedbackConstants.LONG_PRESS, false);
        sendCloseSystemWindows(SYSTEM_DIALOG_REASON_GLOBAL_ACTIONS);
        mWindowManagerFuncs.shutdown(behavior == LONG_PRESS_POWER_SHUT_OFF);
        break;
    }
}
```

STEP4(showGlobalActionsInternal)

PhoneWindowManager.java->showGlobalActionsInternal

```
void showGlobalActionsInternal() {
    // 关闭系统dialogs
    sendCloseSystemWindows(SYSTEM_DIALOG_REASON_GLOBAL_ACTIONS);
    if (mGlobalActions == null) {
        // 创建GlobalActions
        mGlobalActions = new GlobalActions(mContext, mWindowManagerFuncs);
    }
    final boolean keyguardShowing = isKeyguardShowingAndNotOccluded();
    // 重点看showDialog
    mGlobalActions.showDialog(keyguardShowing, isDeviceProvisioned());
    if (keyguardShowing) {
        // since it took two seconds of long press to bring this up,
        // poke the wake lock so they have some time to see the dialog.
        mPowerManager.userActivity(SystemClock.uptimeMillis(), false);
    }
}
```

首先调用sendCloseSystemWindows函数，发送由于全局关机动作的原因，最终会调用ActivityManagerService类的closeSystemDialogs函数关闭其他的系统对话框。利用单例模式创建GlobalActions对象，并保存到其成员变量mGlobalActions中，最终会调用GlobalActions的showDialog方法进行显示关机对话框。

STEP5(showDialog)

```
/**
 * Show the global actions dialog (creating if necessary)
 * @param keyguardShowing True if keyguard is showing
 */
public void showDialog(boolean keyguardShowing, boolean isDeviceProvisioned) {
    mKeyguardShowing = keyguardShowing;
    mDeviceProvisioned = isDeviceProvisioned;
    if (mDialog != null) {
        mDialog.dismiss();
        mDialog = null;
        // Show delayed, so that the dismiss of the previous dialog completes
        mHandler.sendEmptyMessage(MESSAGE_SHOW);
    } else {
        handleShow();
    }
    // 最后都会调用handleShow()
}
```

```
private void handleShow() {
    awakenIfNecessary();
    mDialog = createDialog();   //创建mDialog对象
    prepareDialog();   //准备dialog
    // If we only have 1 item and it's a simple press action, just do this action.
    if (mAdapter.getCount() == 1
            && mAdapter.getItem(0) instanceof SinglePressAction
            && !(mAdapter.getItem(0) instanceof LongPressAction)) {
        ((SinglePressAction) mAdapter.getItem(0)).onPress();    //调用onPress函数
    } else {
        WindowManager.LayoutParams attrs = mDialog.getWindow().getAttributes();
        attrs.setTitle("GlobalActions");
        mDialog.getWindow().setAttributes(attrs);
        mDialog.show();     //显示dialog
        mDialog.getWindow().getDecorView().setSystemUiVisibility(View.STATUS_BAR_DISABLE_EX
    }
}
```

判断是否会显示keyguard

- 如果之前已经有全局对话框显示，则发生延迟消息，以便其显示完后最终关闭
- 如果是第一次启动全局对话框，则会进入handleShow方法中创建全局对话框，将关机选择、重启选择、飞行模式选择以封装的Action对象添加在适配器列表中。

封装完成Action后就是创建关机对话框，采用MyAdapter适配器保存这些匹配。

STEP6(onPress)

```
@Override
public void onPress() {
    // shutdown by making sure radio and power are handled accordingly.
    mWindowManagerFuncs.shutdown(false /* confirm */);    //关机
}
```

在创建全局对话框的同时会对每个选项绑定事件,如果是短按关机Aciton会进入到PowerAction的onPress函数，进入到mWindowManagerFuncs.shutDown方法进行处理。

```
@Override
public void shutdown(boolean confirm) {
    ShutdownThread.shutdown(mContext, PowerManager.SHUTDOWN_USER_REQUESTED, confirm);
}
```

在PhoneWindowManager的初始化过程可知，mWindowManagerFuncs被赋值为WindowManagerService，因此会调用WindowManagerService的shutdown方法。 最终会调用到ShutdownThread中的shutdownInner方法。

STEP7(shutdownInner)

```
public static void shutdown(final Context context, String reason, boolean confirm) {
    mReboot = false;
    mRebootSafeMode = false;   //不是重启
    mReason = reason;   //记录关机的原因
    shutdownInner(context, confirm);
}
static void shutdownInner(final Context context, boolean confirm) {
    // ensure that only one thread is trying to power down.
    // any additional calls are just returned
    synchronized (sIsStartedGuard) {   //保证只有一个关机线程
        if (sIsStarted) {
            Log.d(TAG, "Request to shutdown already running, returning.");
            return;
        }
    }
    final int longPressBehavior = context.getResources().getInteger(
                    com.android.internal.R.integer.config_longPressOnPowerBehavior);   //获
    final int resourceId = mRebootSafeMode
            ? com.android.internal.R.string.reboot_safemode_confirm
            : (longPressBehavior == 2         //关机确认信息
                    ? com.android.internal.R.string.shutdown_confirm_question
                    : com.android.internal.R.string.shutdown_confirm);
    Log.d(TAG, "Notifying thread to start shutdown longPressBehavior=" + longPressBehavior)
    if (confirm) {
        final CloseDialogReceiver closer = new CloseDialogReceiver(context);
        if (sConfirmDialog != null) {
            sConfirmDialog.dismiss();
        }
        sConfirmDialog = new AlertDialog.Builder(context)//需要再次确认关机, 创建dialog
                .setTitle(mRebootSafeMode
                        ? com.android.internal.R.string.reboot_safemode_title
                        : com.android.internal.R.string.power_off)
                .setMessage(resourceId)
                .setPositiveButton(com.android.internal.R.string.yes, new DialogInterface.O
                    public void onClick(DialogInterface dialog, int which) {
                        beginShutdownSequence(context);     //确认关机
                    }
                })
                .setNegativeButton(com.android.internal.R.string.no, null)  //不关机了
                .create();
        closer.dialog = sConfirmDialog;
        sConfirmDialog.setOnDismissListener(closer);
        sConfirmDialog.getWindow().setType(WindowManager.LayoutParams.TYPE_KEYGUARD_DIALOG)
        sConfirmDialog.show();
    } else {
        beginShutdownSequence(context);   //如果不需要确认就直接关机
    }
}
```

通过传进来的confirm参数来判断是否要显示关机对话框

- 如果显示会设置关键对话框
- 如果不显示直接关机

无论显示与否，最后会进入到beginShutdownSequence方法，做进一步的关机处理。

STEP8(beginShutdownSequence)

- 准备关机铃声和动画
- 申请电源锁保持屏幕亮屏
- 运行thread初始化关机

STEP9(shutdownThread.run)

1. 发送关机广播通知所有注册该广播的程序进行关机处理
2. 关闭Am
3. 关闭pm
4. 关闭sm
5. 调用rebootOrShutdown处理关机/重启

详见下图

不管是重启还是关机，最后都是使用SystemProperties写系统属性的方式来实现的。可以用命令adb shell setprop sys.powerctl shutdown来执行关机。

***注 shutdownThread.run方法在android9中有些许变化。**

STEP10

进入到PowerManagerServie的lowLevelShutdown方法，进入到SystemProperties.set方法，将“sys.powerctl”属性设置为“shutdown”最后进入native层完成关机操作。

***注 在关机之前获得振动器震动，由于震动与调用power关机异步进行，所以为了避免关机太快来不及震动，上层会在调用用振动器后睡500ms。最后才经过内核进行关机。**

Native层

采用property_set函数来设置系统属性的值，通过init进程解析init.rc文件的生成的动作的列表，根据传进去的属性值(“shutdown”)，最终调用do_powerctl函数做关机的操作