文章主要记录一些比较正要的变量及函数，仅供自己参考。

http://www.aichengxu.com/other/2127382.htm

PhoneWindowManager是一个策略类，WMS.mPolicy就是该类的对象。

1、mSafeMode

安全模式。

2、mSystemReady

framework层中很多服务都有mSystemReady这个同名变量，一般在systemReady()函数中设为true，表示当前服务启动完毕，可以正常使用了。

3、mSystemBooted

AMS.ensureBootCompleted()-->AMS.enableScreenAfterBoot()-->WindowManagerService.enableScreenAfterBoot()-->systemBooted()-->mSystemBooted=true;

4、mLidState、mLidOpenRotation

这两个变量跟滑盖硬件手机有关，前一个表示滑盖的状态，后一个表示滑盖方向。

5、mUserRotationMode

旋转模式，包含两种：USER\_ROTATION\_FREE、USER\_ROTATION\_LOCKED，用户在设置中可进行选择。USER\_ROTATION\_FREE即允许屏幕旋转；USER\_ROTATION\_LOCKED，关闭屏幕旋转。

WMS.freezeRotation()-->PhoneWindowManager.setUserRotationMode(WindowManagerPolicy.USER\_ROTATION\_LOCKED,rotation)-->Settings.System.putIntForUser()

WMS.thawRotation()-->PhoneWindowManager.setUserRotationMode(WindowManagerPolicy.USER\_ROTATION\_FREE)-->Settings.System.putIntForUser()

6、mUserRotation

用户设定的屏幕方向，比如在横屏状态下关闭自动旋屏，那么mUserRotation=横屏；在竖屏状态下关闭自动旋屏，那么mUserRotation=竖屏。PhoneWindowManager.updateSettings()函数中会根据数据库字段更新这个变量值。

7、mAllowAllRotations

不知道干嘛的。

8、mCurrentAppOrientation

从名字可以知道表示当前App的旋转方向。mCurrentAppOrientation的赋值流程如下：WMS.updateOrientationFromAppTokensLocked()-->PhoneWindowManager.setCurrentOrientationLw()-->mCurrentAppOrientation = newOrientation;-->updateOrientationListenerLp()-->needSensorRunningLp().

<span style="font-size:18px;"> void updateOrientationListenerLp() {

if (!mOrientationListener.canDetectOrientation()) {

return;

}

boolean disable = true;

if (mScreenOnEarly) {

if (needSensorRunningLp()) {

disable = false;

//enable listener if not already enabled

if (!mOrientationSensorEnabled) {

mOrientationListener.<span style="background-color: rgb(255, 102, 102);">enable</span>();

if(localLOGV) Slog.v(TAG, "Enabling listeners");

<span style="background-color: rgb(255, 153, 102);">mOrientationSensorEnabled </span>= true;

}

}

}

if (disable && mOrientationSensorEnabled) {

mOrientationListener.<span style="background-color: rgb(255, 102, 102);">disable</span>();

<span style="background-color: rgb(255, 153, 102);">mOrientationSensorEnabled </span>= false;

}

}

</span>

从上述代码调用来看，当应用窗口规定了屏幕方向时（android：screenoriental），这个方向值最终会保存在mCurrentAppOrientation中，同时调用updateOrientationListenerLp()来关闭或开启方向监听函数。如果在设置中关闭了自动旋屏，那么needSensorRunningLp()就会返回false，否则返回true，如果应用规定的屏幕方向，needSensorRunningLp()也会返回true。updateOrientationListenerLp()函数中根据needSensorRunningLp()返回的值来决定使能mOrientationListener。在enable()和disable()函数中只是简单调用mSensorManager.registerListener()和mSensorManager.unregisterListener()来注册和注销监听函数。

9、mLongPressOnPowerBehavior

mLongPressOnPowerBehavior在未长按过电源键是值为-1，长按电源键后值为1.

10、mHasSoftInput

是否有软键盘。目前的手机只要安装了输入法，这个变量值变为true。该值也在updateSettings()进行更新。

11、mScreenOnEarly、mScreenOnFully

①Notifier.sendNextBroadcast()-->Notifier.sendGoToSleepBroadcast()-->PhoneWindowManager.screenTurnedOff()-->mScreenOnEarly = false;mScreenOnFully=false;

②Notifier.sendNextBroadcast()-->Notifier.sendWakeUpBroadcast()-->PhoneWindowManager.screenTurnedOn()-->mScreenOnEarly = true;

③screenTurningOn()-->waitForKeyguard()-->waitForKeyguardWindowDrawn()-->finishScreenTurningOn()-->mScreenOnFully=true;

由①②③可以看出，mScreenOnEarly、mScreenOnFully都是表示亮屏，只不过之有一点细微差别。

12、mOrientationSensorEnabled

表示方向传感器是否打开。其实并不存在打开和关闭方向传感器之说，这点看第8条中mOrientationSensorEnabled的赋值便知道了，这个只是用来表示是否有注册传感器监听。

13、mOverscanScreenLeft、mOverscanScreenTop、mOverscanScreenWidth、mOverscanScreenHeight

没研究明白干什么的，不过就是表示“The current size of the screen”.

**14、mOverscanLeft、mOverscanTop、mOverscanRight、mOverscanBottom**

表示overscan的矩形区域。

在命令行下输入wm overscan 0,0,360,640后，手机显示屏变小，显示在【0,0,360,640】区域，mOverscanLeft、mOverscanTop、mOverscanRight、mOverscanBottom的值分别为[0,0,360,640]。

wm overscan 0,0,360,640-->Wm.runDisplayOverscan()-->WMS.setOverscan()-->WMS.setOverscanLocked()-->WMS.reconfigureDisplayLocked()-->WMS.configureDisplayPolicyLocked()-->PhoneWindowManager.setDisplayOverscan()

<span style="font-size:18px;"> public void setDisplayOverscan(Display display, int left, int top, int right, int bottom) {

if (display.getDisplayId() == Display.DEFAULT\_DISPLAY) {

mOverscanLeft = left;

mOverscanTop = top;

mOverscanRight = right;

mOverscanBottom = bottom;

}

}

</span>

wm是高通平台（MTK平台类似的命令为am,其他平台没接触，不太清楚）下对手机分辨率、像素密度、显示区域进行设置的命令。

15、mRestrictedOverscanScreenLeft、mRestrictedOverscanScreenTop、mRestrictedOverscanScreenWidth、mRestrictedOverscanScreenHeight

在第14条输入wm overscan 0,0,360,640命令后的情况下这几个值仍然为[0,0,720,1280]

<span style="font-size:18px;">beginLayoutLw(){

................

mOverscanScreenLeft = mRestrictedOverscanScreenLeft = 0;

mOverscanScreenTop = mRestrictedOverscanScreenTop = 0;

mOverscanScreenWidth = mRestrictedOverscanScreenWidth = displayWidth;

mOverscanScreenHeight = mRestrictedOverscanScreenHeight = displayHeight;

..........

}</span>

16、mUnrestrictedScreenLeft、mUnrestrictedScreenTop、mUnrestrictedScreenWidth、mUnrestrictedScreenHeight

在第14条输入wm overscan 0,0,360,640命令后的情况下这几个值为[0,0,360,640]

17、mSystemLeft、mSystemTop、mSystemRight、mSystemBottom

<span style="font-size:18px;"> layoutWindowLw(){

............

// Default policy decor for the default display

dcf.left = mSystemLeft;

dcf.top = mSystemTop;

dcf.right = mSystemRight;

dcf.bottom = mSystemBottom;

.........

}</span>

PhoneWindowManager.layoutWindowLw()-->WindowState.computeFrameLw()-->WindowState.mDecorFrame.set(dcf);

由上述流程知道每个WindowState.mDecorFrame数值来源为PhoneWindowManager中的mSystemLeft、mSystemTop、mSystemRight、mSystemBottom。

18、mDockLayer、mStatusBarLayer

mDockLayer保存的是输入法窗口的Z轴高度。mStatusBarLayer中保存的是状态栏窗口的Z轴高度。

19、mStatusBar、mKeyguard、mNavigationBar

分别表示状态栏窗口、keyguard窗口、导航栏窗口。

WMS.addWindow()-->PhoneWindowManager.prepareAddWindowLw()-->mKeyguard = win;

20、mFocusedWindow

mFocusedWindow 中保存的是焦点窗口。WMS.updateFocusedWindowLocked()-->PhoneWindowManager.focusChangedLw()-->mFocusedWindow = newFocus；

updateSystemUiVisibilityLw()-->updateSystemBarsLw()

21、mFocusedApp

mFocusedApp表示属于哪个Activity。 mFocusedApp = win.getAppToken();

<span style="font-size:18px;"> public IApplicationToken getAppToken() {

return mAppToken != null ? mAppToken.appToken : null;

}</span>

如果该WindowState是一个Activity窗口，那么就返回mAppToken.appToken，否则返回null。

22、mWinDismissingKeyguard

解除锁屏的窗口。比如来电后会先解锁keyguard，那么phone窗口就会保存在mWinDismissingKeyguard中，phone窗口为什么能解锁keyguard，是因为phone窗口中设置了FLAG\_DISMISS\_KEYGUARD属性。

23、mForcingShowNavBar、mForcingShowNavBarLayer

表示是否强制显示导航栏。mForcingShowNavBar在applyPostLayoutPolicyLw()函数中赋值：

<span style="font-size:18px;"> if (mTopFullscreenOpaqueWindowState == null && (win.getAttrs().privateFlags

&WindowManager.LayoutParams.PRIVATE\_FLAG\_FORCE\_SHOW\_NAV\_BAR) != 0

|| (win.isVisibleLw() && attrs.type == TYPE\_INPUT\_METHOD)) {

if (mForcingShowNavBarLayer < 0) {

mForcingShowNavBar = true;

mForcingShowNavBarLayer = win.getSurfaceLayer();

}

}</span>

如果上面的窗口设定了PRIVATE\_FLAG\_FORCE\_SHOW\_NAV\_BAR属性或者当前窗口是输入法窗口，那么就把mForcingShowNavBar设为true，mForcingShowNavBarLayer就保存着当前窗口的Z轴高度值。

24、beginPostLayoutPolicyLw()、applyPostLayoutPolicyLw()、finishPostLayoutPolicyLw()

这三个函数在performLayoutAndPlaceSurfacesLockedInner()中被调用，主要用来在窗口layout后将窗口的一些信息post到PhoneWindowManager中。

private final void performLayoutAndPlaceSurfacesLockedInner(boolean recoveringMemory) {

…………….

// FIRST LOOP: Perform a layout, if needed.

if (repeats < 4) {

<span style="background-color: rgb(255, 153, 102);">performLayoutLockedInner</span>(displayContent, repeats == 1,

false /\*updateInputWindows\*/);

} else {

Slog.w(TAG, "Layout repeat skipped after too many iterations");

}

// FIRST AND ONE HALF LOOP: Make WindowManagerPolicy think

// it is animating.

displayContent.pendingLayoutChanges = 0;

if (DEBUG\_LAYOUT\_REPEATS) debugLayoutRepeats("loop number "

+ mLayoutRepeatCount, displayContent.pendingLayoutChanges);

if (isDefaultDisplay) {

mPolicy.<span style="background-color: rgb(255, 153, 102);">beginPostLayoutPolicyLw</span>(dw, dh);

for (i = windows.size() - 1; i >= 0; i--) {

WindowState w = windows.get(i);

if (w.mHasSurface) {

mPolicy.<span style="background-color: rgb(255, 153, 102);">applyPostLayoutPolicyLw</span>(w, w.mAttrs);

}

}

displayContent.pendingLayoutChanges |= mPolicy.<span style="background-color: rgb(255, 153, 102);">finishPostLayoutPolicyLw</span>();

………………

}

哪些信息需要post到PhoneWindowManager中呢？看看beginPostLayoutPolicyLw()就知道了。

public void beginPostLayoutPolicyLw(int displayWidth, int displayHeight) {

mTopFullscreenOpaqueWindowState = null;

mForceStatusBar = false;

mForceStatusBarFromKeyguard = false;

mForcingShowNavBar = false;

mForcingShowNavBarLayer = -1;

mHideLockScreen = false;

mAllowLockscreenWhenOn = false;

mDismissKeyguard = DISMISS\_KEYGUARD\_NONE;

mShowingLockscreen = false;

mShowingDream = false;

}

beginPostLayoutPolicyLw()函数主要把要post的信息都初始化了。这些变量将在applyPostLayoutPolicyLw()、finishPostLayoutPolicyLw()中重新被赋值。

①mTopFullscreenOpaqueWindowState保存的是顶层全屏不透明窗口，这个全屏有点歧义，并不是真正的全屏的意思，比如打开短信界面，也认为是全屏的窗口。

②mForceStatusBar表示是否强制显示StatusBar，对于非全屏应用窗口且非keyguard窗口时，mForceStatusBar设为true，这个逻辑很容易理解。如果mForceStatusBar==true，在finishPostLayoutPolicyLw()时便会调用mStatusBarController.setBarShowingLw(true)来显示状态栏。

③mForceStatusBarFromKeyguard，这个没搞懂

④mForcingShowNavBar表示强制显示导航栏，这个在第23条研究过了。

⑤mForcingShowNavBarLayer这个也研究过了，见第23条。

⑥mHideLockScreen，当顶层全屏不透明窗口设置了FLAG\_SHOW\_WHEN\_LOCKED属性时，该变量为true，看下FLAG\_SHOW\_WHEN\_LOCKED属性的解释：

/\*\* Window flag: special flag to let windows be shown when the screen

\* is locked. This will let application windows take precedence over

\* key guard or any other lock screens. Can be used with

\* {@link #FLAG\_KEEP\_SCREEN\_ON} to turn screen on and display windows

\* directly before showing the key guard window. Can be used with

\* {@link #FLAG\_DISMISS\_KEYGUARD} to automatically fully dismisss

\* non-secure keyguards. This flag only applies to the top-most

\* full-screen window.

\*/

比如小米手机在锁屏界面长按home键调出手电筒界面，这个界面应该是设置了FLAG\_SHOW\_WHEN\_LOCKED属性的。注意上面的最后一句话，这个属性只适应于完全全屏的应用窗口。mHideLockScreen的含义也就非常明白了，就是隐藏锁屏窗口，即keyguard窗口。在finishPostLayoutPolicyLw()函数中就会根据mHideLockScreen为true来调用mKeyguard.hideLw(true)来隐藏keyguard。

if (mHideLockScreen) {

if (mKeyguard.hideLw(true)) {

changes |= FINISH\_LAYOUT\_REDO\_LAYOUT

| FINISH\_LAYOUT\_REDO\_CONFIG

| FINISH\_LAYOUT\_REDO\_WALLPAPER;

}

if (!mShowingDream) {

mKeyguardDelegate.setHidden(true);

}

}

⑦mAllowLockscreenWhenOn，顶层全屏应用窗口的属性设置了FLAG\_ALLOW\_LOCK\_WHILE\_SCREEN\_ON，那么mAllowLockscreenWhenOn就会设为true。干嘛用的没看懂。

⑧mDismissKeyguard，保存着keyguard的解锁状态。有三个状态DISMISS\_KEYGUARD\_NONE、DISMISS\_KEYGUARD\_START、DISMISS\_KEYGUARD\_CONTINUE。

if ((attrs.flags & FLAG\_DISMISS\_KEYGUARD) != 0

&& mDismissKeyguard == DISMISS\_KEYGUARD\_NONE) {

if (DEBUG\_LAYOUT) Slog.v(TAG, "Setting mDismissKeyguard true by win " + win);

mDismissKeyguard = mWinDismissingKeyguard == win ?

DISMISS\_KEYGUARD\_CONTINUE : DISMISS\_KEYGUARD\_START;

mWinDismissingKeyguard = win;

mForceStatusBarFromKeyguard = mShowingLockscreen && isKeyguardSecure();

}

⑨mShowingLockscreen，表示顶层全屏应用窗口是正是keyguard，即正在显示锁屏界面。

25、mHomePressed

表示home键是否按下。在interceptKeyBeforeDispatching()中更新值。

26、interceptKeyBeforeDispatching()

interceptKeyBeforeDispatching()函数调用流程：InputDispatcher.dispatchKeyLocked()-->InputDispatcher.doInterceptKeyBeforeDispatchingLockedInterruptible()-->NativeInputManager.interceptKeyBeforeDispatching()-->InputManagerService.interceptKeyBeforeDispatching()-->

InputMonitor.interceptKeyBeforeDispatching()-->PhoneWindowManager.interceptKeyBeforeDispatching()。

interceptKeyBeforeDispatching()函数返回-1表示已经消费掉这个按键事件，返回0表示该案件事件需要按正常的流程通过InputDispatcher发送给对应应用窗口。

27、mTopIsFullscreen

mTopIsFullscreen表示顶层全屏不透明窗口是否是真正的全屏应用窗口。

topIsFullscreen = (lp.flags & WindowManager.LayoutParams.FLAG\_FULLSCREEN) != 0

|| (mLastSystemUiFlags & View.SYSTEM\_UI\_FLAG\_FULLSCREEN) != 0;

lp=mTopFullscreenOpaqueWindowState.getAttrs()；

28、mLockScreenTimeout

mLockScreenTimeout中保存着无操作X秒灭屏时长。比如设置中设置无操作5分钟后灭屏，那么mLockScreenTimeout中就保存着5\*60\*1000ms。

29、mStatusBarController、mStatusBar

mStatusBarController状态栏控制对象，mStatusBar状态栏窗口。

30、mNavigationBarController、mNavigationBar

mNavigationBarController导航栏控制对象，mNavigationBar导航栏窗口。