# 原 Android窗口管理服务WindowManagerService对输入法窗口（Input Method Windo 的管理分析

2013年01月28日 00:57:09 罗升阳 阅读数：42286 更多



在Android系统中，输入法窗口是一种特殊类型的窗口，它总是位于需要使用输入法的窗口的上面。也就是说，一旦WindowManagerS e服务检测到焦点窗口需要使用输入法，那么它就会调整输入法窗口在窗口堆栈中的位置，使得输入法窗口位于在焦点窗口的上面，这样用户可以通过输入法窗口 入字母或者文字。本文就将详细分析WindowManagerService服务是如何管理系统中的输入法窗口的。



在Android系统中，除了输入法窗口之外，还有一种窗口称为输入法对话框，它们总是位于输入窗口的上面。Activity窗口、输入法窗口 输入法对话框的位置关系如图1所示：

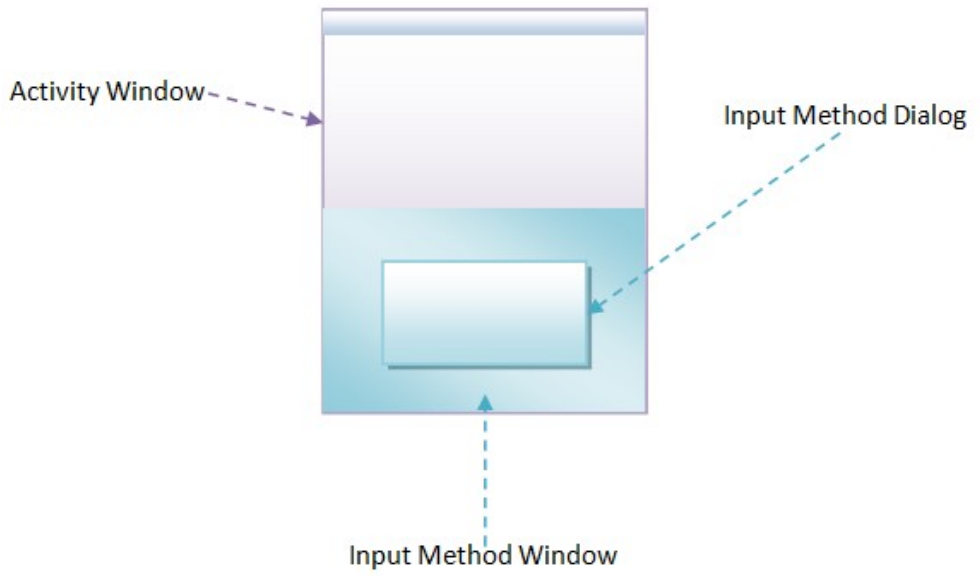


图1 Activity窗口、输入法窗口和输入法对话框的位置关系

在前面Android窗口管理服务WindowManagerService组织窗口的方式分析一文中提到，WindowManagerService服务是使用堆栈来组织系统中的窗口的，因此，如果我们在窗口堆栈中观察Activity窗口、输入法窗口和输入法对话框，它们的位置关系就如图2所示：

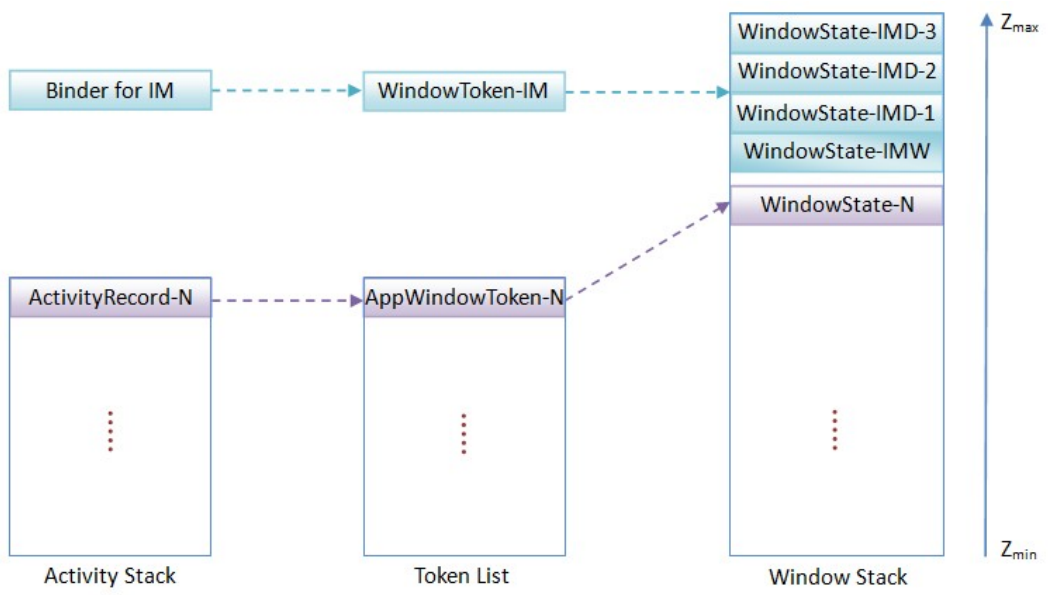


图2 Activity窗口、输入法窗口和输入法对话框在窗口堆栈中的位置关系

图2中的对象的关系如下所示：

1. 在ActivityManagerService服务内部的Activity组件堆栈顶端的ActivityRecord对象N描述的是系统当前激活的Activity组件。

2. ActivityRecord对象N在WindowManagerService服务内部的窗口令牌列表顶端对应有一个AppWindowToken对象N。

3. AppWindowToken对象N在WindowManagerService服务内部的窗口堆栈中对应有一个WindowState对象N，用来描述系统当前激活的Activity组件窗口。

4. WindowState对象N上面有一个WindowState对象IMW，用来描述系统中的输入法窗口。

5. WindowState对象IMW上面有三个WindowState对象IMD-1、IMD-2和IMD-3，它们用来描述系统中的输入法对话框。

6. 系统中的输入法窗口以及输入法对话框在WindowManagerService服务内部中对应的窗口令牌是由WindowToken对象IM来描述的。

7. WindowToken对象IM在InputMethodManagerService服务中对应有一个Binder对象。

总的来说，就是图2描述了系统当前激活的Activity窗口上面显示输入法窗口，而输入法窗口上面又有一系列的输入法对话框的情景。WindowManagerService服务的职能之一就是要时刻关注系统中是否有窗口需要使用输入法。WindowManagerService服务一旦发现有窗口需要使用输入法，那么就会调整输入法窗口以及输入法对话框在窗口堆栈中的位置，使得它们放置在需要使用输入法的窗口的上面。

第一个需要调整输入法窗口以及输入法对话框在窗口堆栈中的位置的情景是增加一个窗口到WindowManagerService服务去的时候。从前面Android应用程序窗口（Activity）与WindowManagerService服务的连接过程分析一文可以知道，增加一个窗口到WindowManagerService服务最终是通过调用WindowManagerService类的成员函数addWindow来实现的。接下来我们就主要分析这个函数中与输入法窗口以及输入法对话框调整相关的[illegible]，如下所示：

```
public class WindowManagerService extends IWindowManager.Stub
        implements Watchdog.Monitor {
    ......

    WindowState mInputMethodWindow = null;
    final ArrayList<WindowState> mInputMethodDialogs = new ArrayList<WindowState>();
    ......

    public int addWindow(Session session, IWindow client,
            WindowManager.LayoutParams attrs, int viewVisibility,
            Rect outContentInsets, InputChannel outInputChannel) {
        ......

        synchronized(mWindowMap) {
            ......

            WindowToken token = mTokenMap.get(attrs.token);
            if (token == null) {
                ......
                if (attrs.type == TYPE_INPUT_METHOD) {
                    ......
                    return WindowManagerImpl.ADD_BAD_APP_TOKEN;
                }
                ......
            }
            ......

            win = new WindowState(session, client, token,
                    attachedWindow, attrs, viewVisibility);
            ......

            boolean imMayMove = true;

            if (attrs.type == TYPE_INPUT_METHOD) {
                mInputMethodWindow = win;
                addInputMethodWindowToListLocked(win);
                imMayMove = false;
            } else if (attrs.type == TYPE_INPUT_METHOD_DIALOG) {
                mInputMethodDialogs.add(win);
                addWindowToListInOrderLocked(win, true);
                adjustInputMethodDialogsLocked();
                imMayMove = false;
            }
            ......

            boolean focusChanged = false;
            if (win.canReceiveKeys()) {
                focusChanged = updateFocusedWindowLocked(UPDATE_FOCUS_WILL_ASSIGN_LAYERS);
                if (focusChanged) {
                    imMayMove = false;
                }
            }

            if (imMayMove) {
                moveInputMethodWindowsIfNeededLocked(false);
            }

            ......
        }

        ......
    }

    ......
}
```

这个函数定义在文件frameworks/base/services/java/com/android/server/WindowManagerService.java中。

如果当前增加到WindowManagerService服务来的是一个输入法窗口，即参数attrs所描述的一个WindowManager.LayoutParams对象的成员变量type的值等于TYPE_INPUT_METHOD，那么就要求与该输入法窗口所对应的类型为WindowToken的窗口令牌已经存在，否则的话，WindowManagerService类的成员函数

InputMethodManagerService服务请求WindowManagerService服务创建的，即调用WindowManagerService类的成员函数addWindowToken来创建的，具体可以参考前面Android窗口管理服务WindowManagerService组织窗口的方式分析一文。

如果当前增加到WindowManagerService服务来的是一个输入法窗口，那么就会将前面为它所创建的一个WindowState对象win保存在WindowManagerService类的成员变量mInputMethodWindow中，接着还会调用WindowManagerService类的成员函数addInputMethodWindowToListLocked来将该WindowState对象插入到窗口堆栈的合适位置去。

如果当前增加到WindowManagerService服务来的是一个输入法对话框，即参数attrs所描述的一个WindowManager.LayoutParams对象的成员变量type的值等于TYPE_INPUT_METHOD_DIALOG，那么就会将前面为它所创建的一个WindowState对象win添加到WindowManagerService类的成员变量mInputMethodDialogs所描述的一个ArrayList中去，并且先后调用WindowManagerService类的成员函数addWindowToListInOrderLocked和adjustInputMethodDialogsLocked来将该WindowState对象插入到窗口堆栈的合适位置去。

在上述两种情况中，由于用来描述输入法窗口或者输入法对话框的WindowState对象已经被插入到了窗口堆栈中的合适位置，因此，接下来不再需要考虑移动该输入法窗口或者输入法对话框了，这时候变量imMayMove的值就会被设置为false。

另一方面，如果当前增加到WindowManagerService服务来的既不是一个输入法窗口，也不是一个输入法对话框，并且该窗口需要接收键盘事件，即前面所创建的WindowState对象win的成员函数canReceiveKeys的返回值为true，那么就可能会导致系统当前获得焦点的窗口发生变化，这时候就需要调用WindowManagerService类的成员函数updateFocusedWindowLocked来重新计算系统当前获得焦点的窗口。如果系统当前获得焦点的窗口发生了变化，那么WindowManagerService类的成员函数updateFocusedWindowLocked的返回值focusChanged就会等于true，同时系统的输入法窗口和输入法对话框在窗口堆栈中的位置也会得到调整，即位它们会位于系统当前获得焦点的窗口的上面，因此，这时候变量imMayMove的值也会被设置为false，表示接下来不再需要考虑移动系统中的输入法窗口或者输入法对话框在窗口堆栈中的位置。

最后，如果变量imMayMove的值保持为初始值，即保持为true，那么就说明当前增加的窗口可能会引发系统的输入法窗口和输入法对话框在窗口堆栈中的位置发生变化，因此，这时候就需要调用WindowManagerService类的成员函数moveInputMethodWindowsIfNeededLocked来作检测，并且在发生变化的情况下，将系统的输入法窗口和输入法对话框移动到窗口堆栈的合适位置上去。

从上面的分析就可以知道，在增加一个窗口的过程中，可能需要调用WindowManagerService类的成员函数addInputMethodWindowToListLocked、addWindowToListInOrderLocked、adjustInputMethodDialogsLocked和moveInputMethodWindowsIfNeededLocked来移动系统的输入法窗口和输入法对话框，其中，WindowManagerService类的成员函数addWindowToListInOrderLocked在前面Android窗口管理服务WindowManagerService组织窗口的方式分析一文已经分析过了，本文只要关注其余三个成员函数的实现。

第二个需要调整输入法窗口以及输入法对话框在窗口堆栈中的位置的情景是一个应用程序进程请求WindowManagerService服务重新布局一个窗口的时候。从前面Android窗口管理服务WindowManagerService计算Activity窗口大小的过程分析一文可以知道，应用程序进程请求WindowManagerService服务重新布局一个窗口最终是通过调用WindowManagerService类的成员函数relayoutWindow来实现的。接下来我们就主要分析这个函数中与输入法窗口以及输入法对话框调整相关的逻辑，如下所示：

```
public class WindowManagerService extends IWindowManager.Stub
        implements Watchdog.Monitor {
    ......

    public int relayoutWindow(Session session, IWindow client,
            WindowManager.LayoutParams attrs, int requestedWidth,
            int requestedHeight, int viewVisibility, boolean insetsPending,
            Rect outFrame, Rect outContentInsets, Rect outVisibleInsets,
            Configuration outConfig, Surface outSurface) {
        ......

        synchronized(mWindowMap) {
            WindowState win = windowForClientLocked(session, client, false);
            ......
            ......

            int attrChanges = 0;
            int flagChanges = 0;
            if (attrs != null) {
                flagChanges = win.mAttrs.flags ^= attrs.flags;
                attrChanges = win.mAttrs.copyFrom(attrs);
            }
            ......

            boolean imMayMove = (flagChanges&(
                    WindowManager.LayoutParams.FLAG_ALT_FOCUSABLE_IM |
                    WindowManager.LayoutParams.FLAG_NOT_FOCUSABLE)) != 0;

            boolean focusMayChange = win.mViewVisibility != viewVisibility
                    || ((flagChanges&WindowManager.LayoutParams.FLAG_NOT_FOCUSABLE) != 0)
                    || (!win.mRelayoutCalled);
            ......

            if (viewVisibility == View.VISIBLE &&
                    (win.mAppToken == null || !win.mAppToken.clientHidden)) {
                displayed = !win.isVisibleLw();
                ......

```

```
                displayed = true;
            }
        ......

        if (win.mAttrs.type == TYPE_INPUT_METHOD
                && mInputMethodWindow == null) {
            mInputMethodWindow = win;
            imMayMove = true;
        }

        if (displayed) {
            focusMayChange = true;
        }

        ......
    } else {
        ......

        if (win.mSurface != null) {
            ......
            // If we are not currently running the exit animation, we
            // need to see about starting one.
            if (!win.mExiting || win.mSurfacePendingDestroy) {
                ......

                if (!win.mSurfacePendingDestroy && win.isWinVisibleLw() &&
                        applyAnimationLocked(win, transit, false)) {
                    focusMayChange = true;
                    win.mExiting = true;
                } else if (win.isAnimating()) {
                    // Currently in a hide animation... turn this into
                    // an exit.
                    win.mExiting = true;
                } else if (win == mWallpaperTarget) {
                    // If the wallpaper is currently behind this
                    // window, we need to change both of them inside
                    // of a transaction to avoid artifacts.
                    win.mExiting = true;
                    win.mAnimating = true;
                } else {
                    if (mInputMethodWindow == win) {
                        mInputMethodWindow = null;
                    }
                    win.destroySurfaceLocked();
                }
            }
        }

        ......
    }

    if (focusMayChange) {
        ......
        if (updateFocusedWindowLocked(UPDATE_FOCUS_WILL_PLACE_SURFACES)) {
            imMayMove = false;
        }
        ......
    }

    // updateFocusedWindowLocked() already assigned layers so we only need to
    // reassign them at this point if the IM window state gets shuffled
    boolean assignLayers = false;

    if (imMayMove) {
        if (moveInputMethodWindowsIfNeededLocked(false) || displayed) {
            // Little hack here -- we -should- be able to rely on the
            // function to return true if the IME has moved and needs
            // its layer recomputed.  However, if the IME was hidden
            // and isn't actually moved in the list, its layer may be
            // out of data so we make sure to recompute it.
            assignLayers = true;
        }
    }
    ......

    if (assignLayers) {
        assignLayersLocked();
    }
```

```
        }
        ......

        return (inTouchMode ? WindowManagerImpl.RELAYOUT_IN_TOUCH_MODE : 0)
                | (displayed ? WindowManagerImpl.RELAYOUT_FIRST_TIME : 0);
    }

    ......
}
```

这个函数定义在文件frameworks/base/services/java/com/android/server/WindowManagerService.java中。

应用程序进程在请求WindowManagerService服务重新布局一个窗口的时候，这个窗口的一些布局参数可能会发生变化，而这些变化可能会同时引发系统的输入法窗口以及输入法对话框在窗口堆栈中的位置发生变化。如果系统的输入法窗口以及输入法对话框在窗口堆栈中的位置发生了变化，那么就需要调整它们在窗口堆栈中的位置。

WindowManagerService类的成员函数relayoutWindow首先调用根据参数session和client来调用另外一个成员函数windowForClientLocked，以便可以获得用来描述要重新布局的窗口的一个WindowState对象win。

WindowState对象win的成员变量mAttrs指向的是一个WindowManager.LayoutParams对象，该WindowManager.LayoutParams对象的成员变量flags描述的是窗口上一次所设置的布局属性标志位，而参数attrs所描述的一个WindowManager.LayoutParams对象的成员变量flags描述的是窗口当前被设置的布局属性标志位。WindowManagerService类的成员函数relayoutWindow通过对这两个标志位执行一个异或操作，就可以知道窗口的哪些布局属性标志位发生了变化，这些变化就记录在变量flagChanges中。

WindowManagerService类的成员函数relayoutWindow在对WindowState对象win所描述的窗口进行布局之前，还要将参数attrs指的是一个WindowManager.LayoutParams对象的内容拷贝到 WindowState对象win的成员变量mAttrs指向的是一个WindowManager.LayoutParams对象中去。在拷贝的过程中，如果发现这两个WindowManager.LayoutParams对象所描述的窗口布局属性有发生变化，那么这些变化就会记录在变量attrChanges中。

在窗口的布局属性标志中，位WindowManager.LayoutParams.FLAG_NOT_FOCUSABLE表示窗口是否可以获得焦点，另外一个位WindowManager.LayoutParams.FLAG_ALT_FOCUSABLE_IM是用来反转WindowManager.LayoutParams.FLAG_NOT_FOCUSABLE位的作用的。一个窗口是否可以获得焦点意味着它是否需要与输入法窗口交互，即如果一个窗口是可以获得焦点的，那么就意味着它需要与输入法窗口交互，否则就不需要。当一个窗口的WindowManager.LayoutParams.FLAG_NOT_FOCUSABLE位等于1，那么就表示窗口不可以获得焦点，即不需要与输入法窗口交互，但是如果该窗口的WindowManager.LayoutParams.FLAG_ALT_FOCUSABLE_IM位也等于1，那么就表示窗口仍然是需要与输入法窗口交互的。另一方面，如果一个窗口的WindowManager.LayoutParams.FLAG_ALT_FOCUSABLE_IM位等于1，但是该窗口的WindowManager.LayoutParams.FLAG_NOT_FOCUSABLE位等于0，那么就表示窗口仍然是不可以与输入法窗口交互的。因此，当前面得到的变量flagChanges的WindowManager.LayoutParams.FLAG_NOT_FOCUSABLE位或者WindowManager.LayoutParams.FLAG_ALT_FOCUSABLE_IM位发生了变化时，都意味着对WindowState对象win所描述的窗口进行重新布局会影响系统中的输入法窗口以及输入法对话框，即该窗口可能会由需要显示输入法窗口以及输入法对话框，到不需要显示输入法窗口以及输入法对话框，反之亦然。最后得到的变量imMayMove的值等于true就表示要移动系统中的输入法窗口以及输入法对话框在窗口堆栈中的位置。

一个窗口由不可获得焦点到可以获得焦点，或者由可获得焦点到不可以获得焦点，即窗口布局属性标志中的WindowManager.LayoutParams.FLAG_NOT_FOCUSABLE位发生了变化，那么就意味着要重新计算系统当前获得焦点的窗口。从前面分析增加窗口到WindowManagerService服务的情景可以知道，当系统当前获得焦点的窗口发生变化时，也意味着需要系统中的移动输入法窗口以及输入法对话框在窗口堆栈中的位置。除了窗口布局属性标志中的WindowManager.LayoutParams.FLAG_NOT_FOCUSABLE位变化会引发系统当前获得焦点的窗口发生变化之外，还有另外两个因素会引发系统当前获得焦点的窗口发生变化。第一个因素是窗口的可见性发生变化。WindowState对象win的成员变量mViewVisibility描述的是窗口上一次布局时的可见性，而参数viewVisibility描述的是窗口当前的可见性，当它们的值不相等时，就意味着窗口的可见性发生了变化。第二个因素是窗口是第一次被应用程序进程请求WindowManagerService服务布局，这时候WindowState对象win的成员变量mRelayoutCalled的值就会等于false。最后得到的变量focusMayChange等于true，就表示需要重新计算系统当前获得焦点的窗口。

WindowState对象win所描述的窗口在此次重新布局中是否会引起移动系统中的输入法窗口以及输入法对话框在窗口堆栈中的位置，还取决于它在的可见性以及它的绘图表面属性等信息，接下来我们就按照 WindowState对象win所描述的窗口当前是可见还是不可见来分别分析。

我们首先分析WindowState对象win所描述的窗口在此次重新布局中是可见的情景，即参数viewVisibility的值等于View.VISIBLE。注意，如果WindowState对象win所描述的是一个Activity窗口，而该Activity组件是不可见的，那么即使参数viewVisibility的值等于View.VISIBLE，那么WindowState对象win所描述的窗口在此次重新布局中也是认为不可见的。从前面Android应用程序窗口（Activity）与WindowManagerService服务的连接过程分析一文可以知道，当WindowState对象win的成员变量mAppToken的值不等于null时，那么该WindowState对象win描述的是一个Activity窗口，而当该成员变量所指向的一个AppWindowToken对象的成员变量clientHidden的值等于false时，就表示对应的Activity组件是可见的。

WindowState对象win所描述的窗口在上一次布局时的可见性可以调用它的成员函数isVisibleLw来获得。如果WindowState对象win所描述的窗口在上一次布局时是不可见的，那么现在就需要将它设置为可见的，即要将它显示出来，这时候变量displayed的值就会等于true。另一方面，如果WindowState对象win所描述的窗口的绘图表面的像素格式发生了变化，即变量attrChanges的WindowManager.LayoutParams.FORMAT_CHANGED位等于1，那么这时候就需要调用WindowState对象win的成员函数destroySurfaceLocked来销毁它所描述的窗口的绘图表面，以便接下来可以为它重新创建一个新的绘图表面，这时候也会将变量displayed的值设置为true，表示接下来是要显示WindowState对象win所描述的窗口的。如果最终得到的变量displayed的值设置为true，那么就相当于说明WindowState对象win所描述的窗口经历一个由不可见到可见的状态变化，因此就可能会导致系统当前获得焦点的窗口发生变化，这时候就会将变量focusMayChange的值设置为true。

如果WindowState对象win描述的是一个输入法窗口，即它的成员变量mAttrs所描述的一个WindowManager.LayoutParams对象的成员变量type的值等于TYPE_INPUT_METHOD，并且系统中的输入法窗口尚未设置，即WindowManagerService类的成员变量mInputMethodWindow的值等于null，那么就说明接下来要显示的其实是输入法窗口，这情况会导致需要移动系统中的输入法窗口以及输入法对话框在窗口堆栈中的位置，因此，这时候除了需要将WindowState对象win保存在WindowManagerService类的成员变量mInputMethodWindow之外，还需要将变量imMayMove的值设置为true。

我们接下来再分析WindowState对象win所描述的窗口在此次重新布局中是不可见的情景，一个窗口变得不可见了，就意味着可能要销毁它的绘图表面，取决于它

如果WindowState对象win的成员变量mExiting等于false时，那么就说明该WindowState对象win所描述的窗口的退出动画可能尚未开始，也可能已经结束。另一方面，如果WindowState对象win的成员变量mSurfacePendingDestroy的值等于true，那么就说明该WindowState对象win所描述的窗口的绘图表面正在等待销毁。这两种情况都需要进一步确定接下来是要开始WindowState对象win所描述的窗口的退出动画，还是要销毁WindowState对象win所描述的窗口的绘图表面。

如果WindowState对象win的成员变量mSurfacePendingDestroy的值等于false，那么同时也意味着它所描述的窗口还未开始显示退出动画，因而它的绘图表面就没有进入正在等待销毁的状态。在这种情况下，如果WindowState对象win所描述的窗口是可见的，即它的成员函数isWinVisibleLw的返回值等于true，那么就意味着要开始该窗口的退出动画了，这是通过调用WindowManagerService类的成员函数applyAnimationLocked来实现的。WindowState对象win所描述的窗口开始退出动画之后，就意味要重新计算系统当前获得焦点的窗口，因此，这时候就会将变量focusMayChange的值设置为true，同时还会将WindowState对象win的成员变量mExiting的值设置为true，表示它描述的窗口正在退出的过程中。

如果WindowState对象win所描述的窗口正在处于退出动画的过程中，即它的成员函数isAnimating的返回值等于true，那么这时候需要将WindowState对象win的成员变量mExiting的值为true。

如果WindowState对象win所描述的窗口已经结束退出动画，但是它仍然是壁纸窗口的目标，即WindowManagerService类的成员变量mLastWallpaperTarget的值不等于null，并且它的值就等于WindowState对象win，那么这时候就需要等待壁纸窗口也退出之后，才销毁WindowState对象win所描述的窗口，因此，这时候就需要将WindowState对象win的成员变量mExiting和mAnimating的值设置为true，即假装它所描述的窗口还处于正在退出的过程，这样做是为了等待壁纸窗口退出完成。

如果WindowState对象win所描述的窗口已经结束退出动画，并且它不是壁纸窗口的目标，那么这时候就需要调用它的成员函数destroySurfaceLocked来销毁它的绘图表面了。在销毁WindowState对象win所描述的窗口之前，还会判断它是否就是系统当前的输入法窗口，即WindowManagerService类的成员变量mInputMethodWindow的值是否等于win。如果等于的话，那么就说明系统当前的输入法窗口被销毁了，因此，就需要将WindowManagerService类的成员变量mInputMethodWindow的值设置为null。

经过上面的一系列操作之后，如果最终得到的变量focusMayChange的值等于true，那么就说明需要重新计算系统当前获得焦点的窗口了，这是通过调用WindowManagerService类的成员函数updateFocusedWindowLocked来实现的。一旦WindowManagerService类的成员函数updateFocusedWindowLocked的返回值为true，那么就说明统当前获得焦点的窗口发生了变化，并且系统中的输入法窗口以及输入法对话框也移动到窗口堆栈中的正确位置了，因此，这时候就会将变量imMayMove的值设置为false。

经过上面的一系列操作之后，如果最终得到的变量imMayMove的值等于true，那么就说明有可能需要移动系统中的输入法窗口以及输入法对话框在窗口堆栈中的位置，这是通过调用WindowManagerService类的成员函数moveInputMethodWindowsIfNeededLocked来实现的。一旦系统中的输入法窗口以及输入法对话框在窗口堆栈中的位置发生了移动，那么WindowManagerService类的成员函数moveInputMethodWindowsIfNeededLocked的返回值就等于true，这时候就需要将变量assignLayers的值设置为true，表示要重新计算系统中的窗口的Z轴位置，以便可以同步到SurfaceFlinger服务中去。注意，如果系统中的输入法窗口以及输入法对话框在窗口堆栈中的位置没有发生变化，但是前面得到的变量displayed的值等于true，那么也是需要将变量assignLayers的值设置为true的，因为这个变量displayed的值等于true意味着WindowState对象win所描述的窗口经历了从不可见到可见的状态变化，因此也需要重新计算系统中的窗口的Z轴位置。

经过上面的一系列操作之后，如果最终得到的变量assignLayers的值等于true，那么就需要调用WindowManagerService类的成员函数assignLayersLocked来执行重新计算统中的窗口的Z轴位置的操作了。在后面的文章中，我们再详细分析WindowManagerService服务是如何计算系统中的窗口的Z轴位置的。

从上面的分析就可以知道，在布局一个窗口的过程中，可能需要调用WindowManagerService类的成员函数moveInputMethodWindowsIfNeededLocked来移动系统的输入法窗口和输入法对话框。再结合前面增加窗口的情景，我们就可以知道，在WindowManagerService类中，与输入法窗口以及输入法对话框相关的成员函数有addInputMethodWindowToListLocked、adjustInputMethodDialogsLocked和moveInputMethodWindowsIfNeededLocked，它们的作用如下所示：

A. 成员函数addInputMethodWindowToListLocked用来将输入法窗口插入到窗口堆栈的合适位置，即插入到需要显示输入法窗口的窗口的上面。

B. 成员函数adjustInputMethodDialogsLocked用来移动输入法对话框到窗口堆栈的合适位置，即移动到输入法窗口的上面。

C. 成员函数moveInputMethodWindowsIfNeededLocked用来检查是否需要移动输入法窗口以及输入法对话框。如果需要的话，那么就将它们移动到窗口堆栈的合适位置去，即将输入法窗口移动到需要显示输入法窗口的窗口的上面，而将输入法对话框移动到输入法窗口的上面。

在分析这三个成员函数的实现之前，我们首先分析WindowManagerService类的成员函数findDesiredInputMethodWindowIndexLocked和moveInputMethodDialogsLocked，它们是两个基本的操作，其中：

D. 成员函数findDesiredInputMethodWindowIndexLocked用来查找输入法窗口在窗口堆栈的正确位置，这个位置刚好就是在需要显示输入法窗口的窗口在窗口堆栈中的上一个位置。

E. 成员函数moveInputMethodDialogsLocked用来将移动输入法对话框移动到输入法窗口的上面去。

接下来我们开始分析上述五个函数的实现。

1. 计算输入法窗口在窗口堆栈中的位置

输入法窗口在窗口堆栈中的位置是通过调用WindowManagerService类的成员函数findDesiredInputMethodWindowIndexLocked来获得的，它首先找到需要显示输入法的窗口在窗口堆栈中的位置，然后再将这个位置加1，就可以得到输入法窗口在窗口堆栈中的位置。

WindowManagerService类的成员函数findDesiredInputMethodWindowIndexLocked定义在文件frameworks/base/services/java/com/android/server/WindowManagerService.java中，它的实现比较长，我们分段来阅读：

```
public class WindowManagerService extends IWindowManager.Stub
        implements Watchdog.Monitor {
    ......

    int findDesiredInputMethodWindowIndexLocked(boolean willMove) {
        final ArrayList<WindowState> localmWindows = mWindows;
```

```
        int i = N;
                10          while (i > 0) {
            i--;
            w = localmWindows.get(i);
            ......

            if (canBeImeTarget(w)) {
                ......

                // Yet more tricksyness!  If this window is a "starting"
                // window, we do actually want to be on top of it, but
                // it is not -really- where input will go.  So if the caller
                // is not actually looking to move the IME, look down below
                // for a real window to target...
                if (!willMove
                        && w.mAttrs.type == WindowManager.LayoutParams.TYPE_APPLICATION_STARTING
                        && i > 0) {
                    WindowState wb = localmWindows.get(i-1);
                    while (i > 1 && wb.mAppToken == w.mAppToken && !canBeImeTarget(wb)) {
                        i--;
                        wb = localmWindows.get(i-1);
                    }
                    if (wb.mAppToken == w.mAppToken && canBeImeTarget(wb)) {
                        i--;
                        w = wb;
                    }
                }
                break;
            }
        }

        mUpcomingInputMethodTarget = w;
```

这段代码从上到下遍历WindowManagerService服务内部的窗口堆栈，即WindowManagerService类的成员变量mWindows所描述的一个ArrayList。如果发现有一个窗口是可见的，并且需要显示输入法窗口，那么整个查找过程就会结束。检查一个窗口是否可见以及需要显示输入法窗口是通过调用WindowManagerService类的成员函数canBeImeTarget来实现的。最后得到的需要显示输入法的窗口就使用WindowState对象w中，这个WindowState对象w接下来还会保存在WindowManagerService类的成员变量mUpcomingInputMethodTarget中，表示它即将要成为输入法窗口的目标窗口。

参数willMove表示调用者计算输入法窗口在窗口堆栈中的位置的目的。如果它的值等于true，那么就说明调用者获得了输入法窗口在窗口堆栈中的位置之后，接下来就会将输入法窗口移动到需要显示输入法窗口的窗口的上面去，否则的话，就说明调用者只是为了知道输入法窗口在窗口堆栈中的位置，而不打算移动输入法窗口。

在从上到下查找需要显示输入法的窗口的过程中，如果找到一个WindowState对象w，它所描述的窗口需要显示输入法窗口，但是这个窗口其实是一个Activity窗口的启动窗口，即该WindowState对象w的成员变量mAttrs所描述的一个WindowManager.LayoutParams对象的成员变量type的值等于WindowManager.LayoutParams.TYPE_APPLICATION_STARTING，那么由于调用WindowManagerService类的成员函数findDesiredInputMethodWindowIndexLocked的目的不是用来移动输入法窗口，而是用来查找输入法窗口在窗口堆栈中的确切位置，因此就不能前面所找到的启动窗口看作是一个需要输入法的窗口，因为这个启动窗口只是Activity窗口在显示过程中出现的一个临时窗口。在这种情况下，这段代码就会继续沿着窗口堆栈往下查找另外一个窗口，该窗口一方面是需要显示输入法窗口的，另一方面要与前面所找到的启动窗口对应的是同一个窗口令牌的。如果能找到这样的一个窗口，那么就会将用来描述它的一个WindowState对象wb保存在变量w中。如果找不到这样的一个窗口，那么这段代码就会继续沿着窗口堆栈往下查找另外一个需要显示输入法的窗口。

我们继续往下阅读代码：

```
        if (willMove && w != null) {
            final WindowState curTarget = mInputMethodTarget;
            if (curTarget != null && curTarget.mAppToken != null) {

                // Now some fun for dealing with window animations that
                // modify the Z order.  We need to look at all windows below
                // the current target that are in this app, finding the highest
                // visible one in layering.
                AppWindowToken token = curTarget.mAppToken;
                WindowState highestTarget = null;
                int highestPos = 0;
                if (token.animating || token.animation != null) {
                    int pos = 0;
                    pos = localmWindows.indexOf(curTarget);
                    while (pos >= 0) {
                        WindowState win = localmWindows.get(pos);
                        if (win.mAppToken != token) {
                            break;
                        }
                        if (!win.mRemoved) {
                            if (highestTarget == null || win.mAnimLayer >
                                    highestTarget.mAnimLayer) {
                                highestTarget = win;
```

```
                    }
                    pos--;
                }
            }

            if (highestTarget != null) {
                ......

                if (mNextAppTransition != WindowManagerPolicy.TRANSIT_UNSET) {
                    // If we are currently setting up for an animation,
                    // hold everything until we can find out what will happen.
                    mInputMethodTargetWaitingAnim = true;
                    mInputMethodTarget = highestTarget;
                    return highestPos + 1;
                } else if (highestTarget.isAnimating() &&
                        highestTarget.mAnimLayer > w.mAnimLayer) {
                    // If the window we are currently targeting is involved
                    // with an animation, and it is on top of the next target
                    // we will be over, then hold off on moving until
                    // that is done.
                    mInputMethodTarget = highestTarget;
                    return highestPos + 1;
                }
            }
        }
    }
```

这段代码用来处理一种特殊情况，即参数willMove的值等于true，并且前面找到了一个需要显示输入法的窗口w，但是当前输入法窗口已经存在一个目标窗口，并且该目标窗口正在切换的过程中。在这种情况下，调用WindowManagerService类的成员函数findDesiredInputMethodWindowIndexLocked的函数就需要等到当前输入法窗口的目标窗口的切换过程结束之后，再将输入法窗口移动到窗口w的上面去，换句话说，就是要保持输入法窗口在它当前的目标窗口的上面，直到它当前的目标窗口的切换过程结束为止。这样WindowManagerService类的成员函数findDesiredInputMethodWindowIndexLocked就需要找到当前输入法窗口的目标窗口在窗口堆栈中的位置，然后再将该位置加1后返回给调用者。

当WindowManagerService类的成员变量mInputMethodTarget的值不等于null，并且它描述的是一个Activity窗口时，即它的成员变量mAppToken的值不等于null时，那么就说明当前输入法窗口已经存在一个目标窗口，而这个目标窗口就是使用WindowManagerService类的成员变量mInputMethodTarget所指向的一个WindowState对象来描述的。接下来这段代码就检查该目标窗口是否正在切换的过程中，即是否正在显示切换动画。如果是的话，那么WindowState对象curTarget的成员变量animating的值就会等于true，或者另外一个成员变量animation的值不等于null，这时候就需要在与该目标窗口所对应的窗口令牌token所描述的一组窗口中，找到一个Z轴位置最大的并且不是已经被移除的窗口。WindowManagerService类的成员函数findDesiredInputMethodWindowIndexLocked的调用者最后就是需要将输入法窗口移动到这个Z轴位置最大的并且不是已经被移除的窗口的上面的。

一个窗口的Z轴位置是记录在用描述它的一个WindowState对象的成员变量mAnimLayer中的，而它是否是已经被移除是记录在这个WindowState对象的成员变量mRemoved中的，因此，如果在窗口令牌token所描述的一组WindowSate对象中，能找到一个WindowSate对象，它的成员变量mAnimLayer的值最大，并且它的成员变量mRemoved不等于true，那么这段代码就会将它保存在变量highestTarget中，并且将它描述的窗口在窗口堆栈中的位置保存在变量highestPos中。

经过前面的一系列计算之后，如果变量highestTarget的值不等于null，那么就说明我们碰到前面所说的特殊的情况，这时候又要分为两种情况来讨论。

第一种情况是当前输入法窗口的目标窗口即将要进入到切换过程，但是这个切换过程尚开始，即WindowManagerService类的成员变量mNextAppTransition的值不等于WindowManagerPolicy.TRANSIT_UNSET。这时候就需要将WindowManagerService类的成员变量mInputMethodTargetWaitingAnim的值设置为true，表示当前输入法窗口的目标窗口正在等待进入切换动画中，并且需要将WindowManagerService类的成员变量mInputMethodTarget修正为变量highestTarget所描述的一个WindowState对象，因为这个WindowState对象才是真正用来描述当前输入法窗口的目标窗口的。

第二种情况是当前输入法窗口的目标窗口已经处于切换的过程了，即变量highestTarget所描述的一个WindowState对象的成员函数isAnimating的返回值为true，并且该目标窗口的Z轴位置大于前面所找到的需要显示输入法窗口的窗口的Z轴，即变量highestTarget所描述的一个WindowState对象的成员变量mAnimLayer的值大于变量w所描述的一个WindowState对象的成员变量mAnimLayer的值。这时候就需要将WindowState对象highestTarget所描述的窗口维持为当前输入法窗口的目标窗口，即将WindowManagerService类的成员变量mInputMethodTarget设置为变量highestTarget，直到WindowState对象highestTarget所描述的窗口的切换过程结束为止。

上述两种情况最后都需要将WindowState对象highestTarget所描述的窗口在窗口堆栈中的位置highestPos加1，然后再返回给调用者，以便调用者接下来可以输入法窗口移动在窗口堆栈的第（highestPos+1）个位置上。

如果我们没有碰到前面所说的特殊的情况，那么WindowManagerService类的成员函数findDesiredInputMethodWindowIndexLocked就会继续往下执行：

```
        if (w != null) {
            if (willMove) {
                ......
                mInputMethodTarget = w;
                if (w.mAppToken != null) {
                    setInputMethodAnimLayerAdjustment(w.mAppToken.animLayerAdjustment);
                } else {
                    setInputMethodAnimLayerAdjustment(0);
                }
            }
            return i+1;
        }
```

作：

A. 将WindowState对象w保存在WindowManagerService类的成员变量mInputMethodTarget中，以便WindowManagerService服务可以知道当前输入法窗口的目标窗口是什么。

B. 检查WindowState对象w描述的窗口是否是Activity窗口，即检查WindowState对象w的成员变量mAppToken的值是否不等于null。如果WindowState对象w描述的窗口是Activity窗口的话，那么就需要根据WindowState对象w的成员变量mAppToken所描述的一个AppWindowToken对象的成员变量animLayerAdjustment来调整系统中的输入法窗口以及输入法对话框的Z轴位置，即在系统中的输入法窗口以及输入法对话框的现有Z轴位置上再增加一个调整量，这个调整量就保存在WindowState对象w的成员变量mAppToken所描述的一个AppWindowToken对象的成员变量animLayerAdjustment中。这个调整的过程是通过调用WindowManagerService类的成员函数setInputMethodAnimLayerAdjustment来实现的。如果WindowState对象w描述的窗口不是Activity窗口，那么就不需要调整系统中的输入法窗口以及输入法对话框的Z轴位置，但是仍然需要调用WindowManagerService类的成员函数setInputMethodAnimLayerAdjustment来将系统中的输入法窗口以及输入法对话框的Z轴位置调整量设置为0，即将WindowManagerService类的成员变量mInputMethodAnimLayerAdjustment的值设置为0。

C. 将变量i的值加1之后返回给调用者，以便调用者可以将系统中的输入法窗口移动到窗口堆栈中的第（i+1）个位置上。

如果变量w的值等于null，那么就说明WindowManagerService类的成员函数findDesiredInputMethodWindowIndexLocked在前面没有找到一个需要显示输入法窗口的窗口，我们继续往下阅读它的代码，以便可以了解它是如何处理这种情况的：

```
        if (willMove) {
            ......
            mInputMethodTarget = null;
            setInputMethodAnimLayerAdjustment(0);
        }
        return -1;
    }

    ......
}
```

WindowManagerService类的成员函数findDesiredInputMethodWindowIndexLocked对在前面没有找到一个需要显示输入法窗口的窗口的情况的处理很简单。它判断参数willMove的值是否等于true。如果等于true的话，那么就会将WindowManagerService类的成员变量mInputMethodTarget的值设置为null，并且调用WindowManagerService类的成员函数setInputMethodAnimLayerAdjustment来将系统中的输入法窗口以及输入法对话框的Z轴位置调整量设置为0。这实际上是用来通知WindowManagerService类的成员函数findDesiredInputMethodWindowIndexLocked的调用者，系统当前没有需要显示输入法窗口的窗口。

最后，WindowManagerService类的成员函数findDesiredInputMethodWindowIndexLocked返回一个-1值给调用者，也是表明系统当前没有需要显示输入法窗口的窗口。

2. 移动输入法对话框移动到输入法窗口的上面

系统中的输入法对话框是需要位于输入法窗口的上面的，因此，我们就需要有一个函数来将输入法对话框移动到输入法窗口的上面去。这个函数就是WindowManagerService类的成员函数moveInputMethodDialogsLocked，它的实现如下所示：

```
public class WindowManagerService extends IWindowManager.Stub
        implements Watchdog.Monitor {
    ......

    void moveInputMethodDialogsLocked(int pos) {
        ArrayList<WindowState> dialogs = mInputMethodDialogs;

        final int N = dialogs.size();
        ......
        for (int i=0; i<N; i++) {
            pos = tmpRemoveWindowLocked(pos, dialogs.get(i));
        }
        ......

        if (pos >= 0) {
            final AppWindowToken targetAppToken = mInputMethodTarget.mAppToken;
            if (pos < mWindows.size()) {
                WindowState wp = mWindows.get(pos);
                if (wp == mInputMethodWindow) {
                    pos++;
                }
            }
            ......
            for (int i=0; i<N; i++) {
                WindowState win = dialogs.get(i);
                win.mTargetAppToken = targetAppToken;
                pos = reAddWindowLocked(pos, win);
            }
            ......
            return;
```

```
            WindowState win = dialogs.get(i);
            reAddWindowToListInOrderLocked(win);
            ......
        }
    }

    ......
}
```

```
            win.mTargetAppToken = null;
```

这个函数定义在文件frameworks/base/services/java/com/android/server/WindowManagerService.java中。

在调用WindowManagerService类的成员函数moveInputMethodDialogsLocked之前，必须要保证系统中的输入法窗口已经被移动到窗口堆栈的正确位置，即已经被移动到需要显示输入法窗口的窗口的上面。这时候参数pos描述的或者是输入法窗口在窗口堆栈中的位置，或者是输入法窗口在窗口堆栈中的位置的上一个位置，即输入法对话框在窗口堆栈中的起始位置。参数pos的值还可以小于0，这时候就表示系统当前没有需要显示输入法窗口的窗口。

在移动输入法对话框到输入法窗口的上面之前，首先要将输入法对话框从窗口堆栈中移除，以便接下来可以重新将它们插入到窗口堆栈中。系统中的输入法对话框都保存在WindowManagerService类的成员变量mInputMethodDialogs所描述的一个ArrayList中，通过调用WindowManagerService类的成员函数来tmpRemoveWindowLocked来移除保存在这个ArrayList中的每一个WindowState对象，就可以将系统中的输入法对话框从窗口堆栈中移除中。注意，将一个WindowState对象从窗口堆栈中移除之后，可能会影响参数pos的值。例如，如果参数pos的值大于被移除的WindowState对象原来在窗口堆栈中的位置值，那么在该WindowState对象被移除之后，参数pos的值就要相应地减少1，这样它才能正确地反映输入法窗口在窗口堆栈中的位置，或者输入法对话框在窗口堆栈中的起始位置。WindowManagerService类的成员函数来tmpRemoveWindowLocked在将一个WindowState对象从窗口堆栈中移除的过程中，会正确处理好参数pos的值，这一点可以参考前面Android窗口管理服务WindowManagerService组织窗口的方式分析一文。

接下来，我们就分为两种情况来分析输入法对话框在窗口是如何移动到输入法窗口的上面去的。

第一种情况是参数pos的值大于等于0，这表明系统当前存在一个需要显示输入法窗口的窗口，这个窗口是通过WindowManagerService类的成员变量mInputMethodTarget所指向的一个WindowState对象来描述的。

前面提到，参数pos描述的或者是输入法窗口在窗口堆栈中的位置，或者是输入法对话框在窗口堆栈中的起始位置，我们首先要将它统一描述为输入法对话框在窗口堆栈中的起始位置。这时候就需要检查保存在窗口堆栈的第pos个位置的WindowState对象wp，是否就是WindowManagerService类的成员变量mInputMethodWindow所指向的那个WindowState对象。如果是的话，那么就说明参数pos描述的或者是输入法窗口在窗口堆栈中的位置，这时候将它的值增加1，就可以让它表示为输入法对话框在窗口堆栈中的起始位置。

得到了输入法对话框在窗口堆栈中的起始位置pos之后，接下来只需要调用WindowManagerService类的成员函数reAddWindowLocked来依次地将保存在WindowManagerService类的成员变量mInputMethodDialogs所描述的一个ArrayList中的第i个WindowState对象保存在窗口堆栈中的第（pos+i）个以位置上即可，这样就可以将输入法对话框都移动到输入法窗口的上面去了。

注意，在移动的过程中，用来描述每一个输入法对话框的每一个WindowState对象的成员变量mTargetAppToken的值设置为WindowManagerService类的成员变量mInputMethodTarget所描述的一个WindowState对象的成员变量mAppToken的值，以便可以将输入法对话框和输入法窗口的目标窗口设置为同一个窗口。

第二种情况是参数pos的值小于0，这表明系统当前不存在一个需要显示输入法窗口的窗口。这时候就需要根据输入法窗口自身的属性来将它们移动到窗口堆栈的合适的位置上去，这是通过调用WindowManagerService类的成员函数reAddWindowToListInOrderLocked来实现的。WindowManagerService类的成员函数reAddWindowToListInOrderLocked的实现可以参考前面Android窗口管理服务WindowManagerService组织窗口的方式分析一文，这里不再详细。

注意，在移动的过程中，用来描述每一个输入法对话框的每一个WindowState对象的成员变量mTargetAppToken的值会被设置为null，这是因为系统当前不存在一个需要显示输入法窗口的窗口，即这时候每一个输入法对话框都没有目标窗口。

理解了WindowManagerService类的成员函数findDesiredInputMethodWindowIndexLocked和moveInputMethodDialogsLocked的实现之后，对WindowManagerService类的另外三个成员函数addInputMethodWindowToListLocked、adjustInputMethodDialogsLocked和moveInputMethodWindowsIfNeededLocked的实现就很有帮助，接下来我们就继续分析这三个成员函数的实现。

3. 插入输入法窗口到需要显示输入法窗口的窗口上面

插入输入法窗口到窗口堆栈的合适位置，使得它位于需要显示输入法窗口的窗口上面，这是通过调用WindowManagerService类的成员函数addInputMethodWindowToListLocked来实现的，它的实现如下所示：

```
public class WindowManagerService extends IWindowManager.Stub
        implements Watchdog.Monitor {
    ......

    void addInputMethodWindowToListLocked(WindowState win) {
        int pos = findDesiredInputMethodWindowIndexLocked(true);
        if (pos >= 0) {
            win.mTargetAppToken = mInputMethodTarget.mAppToken;
            ......
            mWindows.add(pos, win);
            mWindowsChanged = true;
            moveInputMethodDialogsLocked(pos+1);
            return;
        }
        win.mTargetAppToken = null;
        addWindowToListInOrderLocked(win, true);
```

```
    ......
}
```

这个函数定义在文件frameworks/base/services/java/com/android/server/WindowManagerService.java中。

参数win描述的是要添加到窗口堆栈中去的输入法窗口。

WindowManagerService类的成员函数addInputMethodWindowToListLocked首先调用另外一个成员函数findDesiredInputMetho lowIndexLocked来计算输入法窗口在窗口堆栈中的位置，并且保存在变量pos。

如果变量pos的值大于等于0，那么就说明WindowManagerService类的成员函数findDesiredInputMethodWindowIndexLocked在窗 栈中找到了一个合适的位置来放置输入法窗口，于是接下来就会参数win所描述的输入法窗口插入在WindowManagerService类的成员变量mWIndows所描述的 栈的第pos个位置上。由于系统中的输入法对话框要保持在输入法窗口的上面，因此，WindowManagerService类的成员函数addInputMethodWindowToLi ked还需要继续调用另外一个成员函数moveInputMethodDialogsLocked来将系统中的输入法对话框在窗口堆栈中的起始位置设置为（pos+1）。

还有一个地方需要注意的是，前面在调用WindowManagerService类的成员函数addInputMethodWindowToListLocked来计算输入法窗口在窗口堆栈中的位置的时候，已经将用来描述需要显示输入法窗口的Activity窗口的一个WindowState对象保存了WindowManagerService类的成员变量mInputMethodTarget中，因此，这里就需要这个WindowState对象的成员变量mAppToken所指向的一个AppWindowToken对象保存在用来描述输入法窗口的WindowState对象的win的成员变量mTargetAppToken中，以便WindowManagerService服务可以知道当前输入法窗口的目标窗口是什么。

如果变量pos的值小于0，那么就说明WindowManagerService类的成员函数findDesiredInputMethodWindowIndexLocked没有找一个需要输入法窗口的窗口，因此，这时候就需要调用另外一个成员函数addWindowToListInOrderLocked来将参数win所描述的输入法窗口插入到窗口堆栈中去。WindowManagerService类的成员函数addWindowToListInOrderLocked会根据要目标窗口所对应的窗口令牌在窗口令牌列表中的位置以及是否在窗口堆栈中存在其它窗口等信息来在窗口堆栈中找到一个合适的前位置来放置目标窗口，它的具体实现可以参考前面Android窗口管理服务WindowManagerService组织窗口的方式分析一文。将参数win所描述的输入法窗口插入到窗口堆栈中去之后，WindowManagerService类的成员函数addInputMethodWindowToListLocked还需要继续调用另外一个成员函数moveInputMethodDialogsLocked来调整系统中的输入法对话框。

注意，在调用WindowManagerService类的成员函数moveInputMethodDialogsLocked的时候，传递进去的参数pos的值等于-1，这时候WindowManagerService类的成员函数moveInputMethodDialogsLocked就不是直接调整输入法对话框在窗口堆栈中的位置的，而是调用另外一个成员函数reAddWindowToListInOrderLocked来调整的。

还有另外一个地方需要注意的是，由于前面在调用WindowManagerService类的成员函数findDesiredInputMethodWindowIndexLocked的时候，没有找到一个需要输入法窗口的窗口，因此，这里就需要将参数win所描述的一个WindowState对象的成员变量mTargetAppToken的值设置为null，以便WindowManagerService服务可以知道当前输入法窗口的没有目标窗口。

4. 调整输入法对话框在窗口堆栈的位置

一旦系统中存在需要显示输入法窗口的窗口，那么就需要系统中的输入法对话框在窗口堆栈中的位置，使得它们放置在输入法窗品的上面，这是通过调用WindowManagerService类的成员函数adjustInputMethodDialogsLocked来实现的，如下所示：

```
public class WindowManagerService extends IWindowManager.Stub
        implements Watchdog.Monitor {
    ......

    void adjustInputMethodDialogsLocked() {
        moveInputMethodDialogsLocked(findDesiredInputMethodWindowIndexLocked(true));
    }

    ......
}
```

这个函数定义在文件frameworks/base/services/java/com/android/server/WindowManagerService.java中。

WindowManagerService类的成员函数adjustInputMethodDialogsLocked的实现很简单，它首先调用成员函数findDesiredInputMethodWindowIndexLocked来找到输入法窗口在窗口堆栈中的位置，然后再调用成员函数moveInputMethodDialogsLocked来将输入法对话框保存在这个位置之上。

5. 调整输入法窗口在窗口堆栈的位置

当系统中的窗口布局发生了变化之后，例如，当前获得焦点的窗口发生了变化，或者新增了一个窗口，那么都可能需要调整输入法窗口在窗口堆栈中的位置，以便它可以痊于需要显示输入法窗口的窗口的上面，这是通过调用WindowManagerService类的成员函数moveInputMethodWindowsIfNeededLocked来实现的，如下所示：

```
public class WindowManagerService extends IWindowManager.Stub
        implements Watchdog.Monitor {
    ......

    boolean moveInputMethodWindowsIfNeededLocked(boolean needAssignLayers) {
        final WindowState imWin = mInputMethodWindow;
        final int DN = mInputMethodDialogs.size();
```

```
11
12          int imPos = findDesiredInputMethodWindowIndexLocked(true);
13      if (imPos >= 0) {
14          // In this case, the input method windows are to be placed
15          // immediately above the window they are targeting.
16
17          // First check to see if the input method windows are already
18          // located here, and contiguous.
19          final int N = mWindows.size();
20          WindowState firstImWin = imPos < N
21                  ? mWindows.get(imPos) : null;
22
23          // Figure out the actual input method window that should be
24          // at the bottom of their stack.
25          WindowState baseImWin = imWin != null
26                  ? imWin : mInputMethodDialogs.get(0);
27          if (baseImWin.mChildWindows.size() > 0) {
28              WindowState cw = baseImWin.mChildWindows.get(0);
29              if (cw.mSubLayer < 0) baseImWin = cw;
30          }
31
32          if (firstImWin == baseImWin) {
33              // The windows haven't moved...  but are they still contiguous?
34              // First find the top IM window.
35              int pos = imPos+1;
36              while (pos < N) {
37                  if (!(mWindows.get(pos)).mIsImWindow) {
38                      break;
39                  }
40                  pos++;
41              }
42              pos++;
43              // Now there should be no more input method windows above.
44              while (pos < N) {
45                  if ((mWindows.get(pos)).mIsImWindow) {
46                      break;
47                  }
48                  pos++;
49              }
50              if (pos >= N) {
51                  // All is good!
52                  return false;
53              }
54          }
55
56          if (imWin != null) {
57              ......
58              imPos = tmpRemoveWindowLocked(imPos, imWin);
59              ......
60              imWin.mTargetAppToken = mInputMethodTarget.mAppToken;
61              reAddWindowLocked(imPos, imWin);
62              ......
63              if (DN > 0) moveInputMethodDialogsLocked(imPos+1);
64          } else {
65              moveInputMethodDialogsLocked(imPos);
66          }
67
68      } else {
69          // In this case, the input method windows go in a fixed layer,
70          // because they aren't currently associated with a focus window.
71
72          if (imWin != null) {
73              ......
74              tmpRemoveWindowLocked(0, imWin);
75              imWin.mTargetAppToken = null;
76              reAddWindowToListInOrderLocked(imWin);
77              ......
78              if (DN > 0) moveInputMethodDialogsLocked(-1);;
79          } else {
80              moveInputMethodDialogsLocked(-1);;
81          }
82
83      }
84
85      if (needAssignLayers) {
86          assignLayersLocked();
87      }
88
```

22

20

```
91  92      ......
93  }
```

这个函数定义在文件frameworks/base/services/java/com/android/server/WindowManagerService.java中。

WindowManagerService类的成员函数moveInputMethodWindowsIfNeededLocked首先检查系统中是否存在输入法窗口和输入法对话框，即检查WindowManagerService类的成员变量mInputMethodWindow的值是否等于null，并且WindowManagerService类的成员变量mInputMethodDialogs所描述的一个ArrayList的大小是否等于0。如果输入法窗口和输入法对话框都不存在的话，那么就不用调整它们在窗口堆栈中的位置了，否则的话，WindowManagerService类的成员变量mInputMethodWindow所指向的一个WindowState对象就会保存在变量imWin中，以便接下来可以通过它来描述系统中的输入法

在输入法窗口或者输入法对话框存在的情况下，WindowManagerService类的成员函数moveInputMethodWindowsIfNeededLocked就会继续调用另外一个成员函数findDesiredInputMethodWindowIndexLocked来找到输入法窗口在窗口堆栈中的位置，并且保存在变量imPos中。注意，变量imPos的值可能大于等于0，也可能等于-1。当变量imPos的值大于等于0的时候，就说明系统当前存在一个窗口需要显示输入法窗口，而当变量imPos的值等于-1的时候，就说明系统当前不存在一个窗口需要显示输入法窗口，或者系统中不存在输入法窗口。接下来我们分两种情况来分析WindowManagerService类的成员函数moveInputMethodWindowsIfNeededLocked的实现。

第一种情况是变量imPos的值可能大于等于0。这时候可能需要调整输入法窗口在窗口堆栈中的位置，也可能不需要调整输入法窗口在窗口堆栈中的位置，取决于输入法窗口的位置是否已经在窗口堆栈的第imPos个位置上，以及是否所有与输入法相关的窗口都连续在放置在窗口堆栈中。

变量firstImWin描述的是当前位于窗口堆栈中Z轴位置最小的与输入法相关的窗口，它是通过变量imPos来获得的。另外一个变量baseImWin描述的是Z轴位置最小的与输入法相关的窗口。如果这两个变量描述的是同一个窗口，那么就说明输入法窗口的位置已经在窗口堆栈的第imPos个位置上，因此，就有可能不需要调整输入法窗品在窗口堆栈中的位置了。接下来我们就描述如何找到这个Z轴位置最小的与输入法相关的窗口。

如果变量imWin的值不等于null，即WindowManagerService类的成员变量mInputMethodWindow的值不等于null，那么它所描述的窗口就是Z轴位置最小的与输入法相关的窗口，否则的话，Z轴位置最小的与输入法相关的窗口就是位于WindowManagerService类的成员变量mInputMethodDialogs所描述的一个ArrayList的第0个位置上的输入法对话框。这一步得到的Z轴位置最小的与输入法相关的窗口就保存在变量baseImWin中。

如果变量baseImWin所描述的窗口有子窗口，即它所指向的一个WindowState对象的成员变量mChildWindows所描述的一个ArrayList的大小大于0。这时候如果用来描述第一个子窗口的WindowState对象的成员变量mSubLayer的值小于0，那么就说明变量baseImWin所描述的窗口在所有与输入法相关的窗口中的Z轴位置还不是最小的，因为在它的下面还存在着Z轴位置更小的子窗口。在这种情况下，变量baseImWin就会指向这个Z轴位置最小的子窗口。

经过上面的一系列计算之后，如果变量firstImWin和变量baseImWin描述的是同一个窗口，那么还需要继续判断所有与输入法相关的窗口都连续在放置在窗口堆栈中。判断的方法如下所示：

(1). 从窗口堆栈的第（imPos + 1）个位置开始往上查找一个非输入法相关的窗口。

(2). 如果第(1)步能在窗口堆栈中大于等于（imPos+1）的位置pos上找到一个非输入法窗口，那么再继续从第pos个位置开始往上查找一个与输入法相关的窗口。

(3). 如果第(2)步能在窗口堆栈中找到一个与输入法相关的窗口，那么就说明所有与输入法相关的窗口不是连续在放置在窗口堆栈中的，因为在它们中间有一个非输入法相关的窗口，否则的话，就说明所有与输入法相关的窗口都是连续在放置在窗口堆栈中的。

在所有与输入法相关的窗口都是连续在放置在窗口堆栈中的情况下，WindowManagerService类的成员函数moveInputMethodWindowsIfNeededLocked就会直接返回一个false值给调用者，表明不需要调整系统中的输入法窗口以及输入法对话框在窗口堆栈中的位置。

在所有与输入法相关的窗口不是连续在放置在窗口堆栈中的情况下，就需要重新调整系统中的输入法窗口以及输入法对话框在窗口堆栈中的位置。这里又需要分两个情景来讨论。

第一个情景是变量imWin的值不等于null，这时候说明系统中存在一个输入法窗口，因此，就需要调整这个输入法窗口在窗口堆栈中的位置。调整的方法很简单：

(1). 调用WindowManagerService类的成员函数tmpRemoveWindowLocked来从窗口堆栈中移除变量imWin所描述的输入法窗口。在移除的过程中，会同时计算输入法窗口在窗口堆栈中的新位置，这个位置还是保存在变量imPos中。

(2). 调用WindowManagerService类的成员函数reAddWindowLocked重新将变量imWin所描述的输入法窗口插入到窗口堆栈的第imPos个位置中。在插入之前，还会将变量imWin所描述的一个WindowState对象的成员变量mTargetAppToken与WindowManagerService类的成员变量mInputMethodTarget所描述的一个WindowState对象的成员变量mAppToken指向同一个AppWindowToken对象，这样WindowManagerService服务就可以知道imWin所描述的输入法窗口的目标窗口是什么。

(3). 如果系统中还存在输入法对话框，那么就调用WindowManagerService类的成员函数moveInputMethodDialogsLocked来将它们放置在第（imPos+1）个位置上，目的是将它们放置在输入法窗口的上面。

第二个情景是变量imWin的值等于null，这时候说明系统中不存在输入法窗口。在这个情景下，系统中肯定会存在输入法对话框，否则的话，WindowManagerService类的成员函数moveInputMethodWindowsIfNeededLocked在前面就会返回了。因此，WindowManagerService类的成员函数moveInputMethodWindowsIfNeededLocked接下来就会直接调用成员函数moveInputMethodDialogsLocked来将系统中的输入法对话框放置在在第imPos个位置上。

第二种情况是变量imPos的值等于-1。这时候说明系统中不存在需要显示输入法窗口的窗口。这里同样也需要分两个情景来分析。

第一个情景是变量imWin的值不等于null，这时候说明系统中存在一个输入法窗口，因此，就需要调整这个输入法窗口在窗口堆栈中的位置。调整的方法与前面第一种情况的第一个情景是类似的。不过由于事先不知道输入法窗口在窗口堆栈中的位置，因此，这里就会调用WindowManagerService类的成员函数reAddWindowToListInOrderLocked和moveInputMethodDialogsLocked来间接地调整输入法窗口和输入法对话框在窗口堆栈中的位置。注意，在调用WindowManagerService类的成员函数moveInputMethodDialogsLocked的时候，传进去的参数为-1。另外一个地方需要注意的是，在WindowManagerService类的成员函数reAddWindowToListInOrderLocked来间接地调整输入法窗口在窗口堆栈中的位置之前，会将量imWin所描述的一个WindowState对象的成员变量mTargetAppToken的值设置为null，这样WindowManagerService服务就可以知道imWin所描述的输入法窗口没有目标窗口

第二情景是变量imWin的值等于null，这时候系统中不存在输入法窗口。这个情景与前面第一种情况的第二个情景也是类似的。由于系统中不存在输入法窗口，因此只需要调用WindowManagerService类的成员函数moveInputMethodDialogsLocked来间接地输入法对话框在窗口堆栈中的位置即可，即以参数-1来调用WindowManagerService类的成员函数moveInputMethodDialogsLocked。

至此，我们就分析完成WindowManagerService服务对输入法窗口的基本操作了。从分析的过程中，我们可以得到以下两个结论：

A. 系统中与输入法相关的窗口有两种，一种是输入法窗口，另一种是输入法对话框。

B. 当Z轴位置最大的窗口需要使用输入法时，输入法窗口就会位于它的上面，而输入法对话框又会位于输入法窗口的上面。

在WindowManagerService服务中，还有一种类型的窗口与输入法窗口类似，它总是与Activity窗口粘在一起。不过，这种类型的窗口是位于Activity窗口的下面，刚好与输入法窗口相反，它就是壁纸窗口（Wallpaper）。在接下来的一篇文章中，我们就将继续分析WindowManagerService服务是如何管理系统中的壁纸窗口的。敬请关注！

**老罗的新浪微博：http://weibo.com/shengyangluo，欢迎关注！**

22

20

想对作者说点什么

布吉刀： updateFocusedWindowLocked can use for no focus window (1年前 #15楼) 0

chinese_android： 请问：什么事输入法对话框？输入法窗口和输入法对话框有什么区别？ (2年前 #14楼) 0

woshihongliu： 您好，最近在做一个功能，就是给android的虚拟按键新加一个按键，功能可以隐藏虚拟按键，原理是点击该按键的时候调用windowmanager.removeview(mNavigationBarView) 但是在输入法界面的时候就出现了bug，虚拟按键是消失了，但是输入法界面没有随之变化，罗老师对此感兴趣吗？ (4年前 #13楼) 0

登录 查看 20 条热评

**Android窗口管理服务WindowManagerService对输入法窗口..._CSDN博客** 11-16

6. 系统中的输入法窗口以及输入法对话框在WindowManagerService服务内部中对应的窗口令牌是由WindowToken对象IM来描述的。 7. WindowToke...

22

**Android窗口管理服务WindowManagerService对窗口的组织方式分析** 3-25

此外,在输入法管理服务InputMethodManagerService中,每一个输入法窗口都对应有一个Binder对象,这个Binder对象在Window管理服务Window...

20

**Android输入法框架系统(上)** 阅读数 2万+

输入法，就是用来输入字符（包括英文，俄文，中文）的工具。输入法你可以看成是一种字符发生器，它将输入数据... 博文 来自： 一叶梧桐

**Android对输入法窗口的管理分析** 阅读数 3049

原文：Android窗口管理服务WindowManagerService对输入法窗口（InputMethodWindow）的管理分析链接：... 博文 来自： xww810319的专栏

**Android窗口管理服务WindowManagerService对输入法窗口..._CSDN博客** 9-24

在Android系统中,输入法窗口是一种特殊类型的窗口,它总是位于需要使用输入法的窗口的上面。也就是说,一旦WindowManagerService服务检测到焦点...

**Android窗口管理服务WindowManagerService对窗口的组织..._CSDN博客** 11-16

与Activity类似,Android系统中的窗口也是以堆栈的形式组织在WindowManagerService服务...此外,在输入法管理服务InputMethodManagerService中,...

**Android输入法框的梳理** 阅读数 2万+

/frameworks/base/services/java/InputMethodManagerService.java这是整个系统当中，一切与输入法有关的地... 博文 来自： 农场老马的专栏

**Android 中InputMethodManager类的用法** 阅读数 2万+

分类： Android知识总结2012-10-2819:36 2245人阅读 评论(0) 收藏 举报JavaEye社区：http://www.iteye.comA... 博文 来自： pi9nc的专栏

**Android窗口管理服务WindowManagerService对窗口的组织..._CSDN博客** 7-11

与Activity类似,Android系统中的窗口也是以堆栈的形式组织在WindowManagerService服务...此外,在输入法管理服务InputMethodManagerService中,...

**Android窗口管理服务WindowManagerService对壁纸窗口(W..._CSDN博客** 11-17

在Android系统中,壁纸窗口和输入法窗口一样,都是一...在前面Android窗口管理服务WindowManagerService组织窗口...Rect outContentInsets, InputC...

**Android6.0 WMS（三） WMS窗口次序** 阅读数 2162

这篇博客我们主要分析下，窗口位置排序的一些原理。一、添加窗口的时候调整窗口位置上篇博客我们分析了WMS... 博文 来自： kc58236582的博客

**InputMethodService详解** 阅读数 3936

InputMethodService类提供了对输入法的标准实现，我们可以对InputMethodService类进行派生和定制，InputM... 博文 来自： wwlwwy89的专栏

**Android窗口管理服务WindowManagerService对壁纸窗口(W..._CSDN博客** 11-3

结合前面Android窗口管理服务WindowManagerService对窗口的组织方式分析和Android窗口管理服务WindowManagerService对输入法窗口(Input ...

**Android窗口管理服务WindowManagerService计算窗口Z轴..._CSDN博客** 5-23

通过前面几篇文章的学习,我们知道了在Android系统中,无论是普通的Activity窗口,还是特殊的输入法窗口和壁纸窗口,它们都是被WindowManagerServic...

**Android6.0 WMS（五） WMS计算Activity窗口大小的过程分析（二）WMS的relayoutWindow** 阅读数 4198

既上一篇博客，这篇我们分析WMS的relayoutWindow函数。relayoutWindow我们先看下relayoutWindow函数p... 博文 来自： kc58236582的博客

**Android6.0 WMS（七） 窗口Z轴位置** 阅读数 2118

通过前面几篇文章的学习，我们知道了在Android系统中，无论是普通的Activity窗口，还是特殊的输入法窗口和壁... 博文 来自： kc58236582的博客

**Android窗口管理服务WindowManagerService对壁纸窗口（Wallpaper Window）的管理分析** 阅读数 4万+

在Android系统中，壁纸窗口和输入法窗口一样，都是一种特殊类型的窗口，而且它们都是喜欢和一个普通的Activit... 博文 来自： 老罗的Android之旅

**仿今日头条的夜间模式** 阅读数 2534

前段时间查资料，为自己的新闻项目做过一个仿今日头条的夜间模式效果，经过一段时间的测试，发现现在效果的状... 博文 来自： 张小贝的博客

**Android窗口管理分析（2）：WindowManagerService窗口管理之Window添加流程** 阅读数 1501

之前分析说过，WindowManagerService只负责窗口管理，并不负责View的绘制跟图层混合，本文就来分析WMS... 博文 来自： happylishang的专栏

**Android窗口管理服务WindowManagerService显示窗口动画的原理分析** 阅读数 656

在前一文中，我们分析了Activity组件的切换过程。从这个过程可以知道，所有参与切换操作的窗口都会被设置切换... 博文 来自： xu_song的专栏

**Android窗口管理服务WindowManagerService对输入法窗口（Input Method Window）的...** 阅读数 20

在Android系统中，输入法窗口是一种特殊类型的窗口，它总是位于需要使用输入法的窗口的上面。也就是说，一旦... 博文 来自： mmdev

**Android窗口系统第三篇---WindowManagerService中窗口的组织方式** 阅读数 1665

登录 注册 ×

## Android WindowManagerService机制分析：窗口的显示层级

阅读数 2010

WindowManagerService（以下简称WMS）是AndroidFramework中一个重要的系统服务，用来管理系统中窗... 博文 来自： u012551754的专栏

## android中Service组件总结

阅读数 525

Server是android四大组件之一，它与Activity非常类似，最大的区别就是Activity在前台运行，主要作用于界面的交... 博文 来自： gjnm820的博客

## Android 输入法在页面中显示方式位置变化

阅读数 1316

近期做项目的时候，使用TabHostFragment类以便做导航页签，但在子Fragment里需要输入文字，这就导致在显示... 博文 来自： 浅颜如梦

## 重写InputMethodService编写一个简单的输入法

阅读数 3895

编写一个简单的输入法： 博文 来自： Ataraxia的专栏

## Android获取输入法高度——输入法与页面布局无缝切换

阅读数 9245

在QQ或者微信的聊天页面，当输入法和表情栏互相切换时，过度非常自然，而且表情栏高度刚好跟输入法一样。个... 博文 来自： 走向远方

## 搜狗输入法最新版本

01-13

最新输入法最新输入法最新输入法最新输入法最新输入法最新输入法最新输入法最新输入法最新输入法最新输入法最新输入法最新输入法最新输... 下载

## 看不到语言栏，输入法无法转换

阅读数 39

如果语言栏显示不出来的话，有以下几个方法：方法1：在任务栏单击鼠标右键，弹出快捷菜单，把鼠标移动到“工... 博文 来自： lzj0470的专栏

## APP输入法测试点

阅读数 333

输入法弹出和隐藏：1、点击输入框弹出输入法2、输入框弹出高度等于输入法弹出高度3、输入框不能移动4、点击输... 博文 来自： Androidd2015的...

## android输入法问题

阅读数 1260

1谷歌输入法 包名：com.google.android.inputmethod.latin 属于第三方输入法 LatinIME软键盘OpenWnnCJK输... 博文 来自： 格物的专栏

## 输入法原理

阅读数 184

Question1:客户端如何与输入法产生联系首先我们要明确的是输入法就是一个Service，无论是百度输入法还是搜狗... 博文 来自： 汪子涵的博客

## inputmanager对按键事件的获取和向上派发

阅读数 2059

在Android系统中，键盘按键事件是由WindowManagerService服务来管理的，然后再以消息的形式来分发给应用... 博文 来自： liyanfei123456的...

## 记录inputManager使用方法

阅读数 233

//通过context获取系统服务InputMethodManagerinput=(InputMethodManager)getContext().getSystemServi... 博文 来自： m0_37261093的博...

## InputManagerService之事件的初始化与分发

阅读数 1028

该篇文章接总章，来详细谈论说明InputManagerService体系，从初始化到事件获取跟分发。咱们在进行前，先明确... 博文 来自： 李云轩的专栏

## Android软键盘(1)---输入法界面管理（打开/关闭/状态获取）

阅读数 2317

一、打开输入法窗口:InputMethodManagerinputMethodManager=(InputMethodManager)getSystemService(... 博文 来自： lovoo的博客

## Android中软键盘InputMethodManager的弹出和隐藏,以及显示和隐藏的监听

阅读数 3701

1.首先设置软键盘的弹出模式,设置在初始化View的前面@OverrideprotectedvoidonCreate(BundlesavedInstance... 博文 来自： zfgrinm的专栏

## WindowManagerService如何管理应用程序窗口

阅读数 358

一、引言如果对于Android窗口视图是怎么显示的还不是很清楚的话，可以看看我上一篇博文Android视图渲染过程... 博文 来自： 粤语伶仃的专栏

## 输入法与窗口交互

阅读数 177

1.我们在项目中经常会遇到软键盘遮挡页面，输入框或者软键盘自动弹出的场景，在Android中Activity给我们提供了... 博文 来自： Android

## Android窗口管理服务WindowManagerService对窗口的组织方式分析

阅读数 637

我们知道，在Android系统中，Activity是以堆栈的形式组织在ActivityManagerService服务中的。与Activity类似，... 博文 来自： xu_song的专栏

## Android解析WindowManagerService（二）WMS的重要成员和Window的添加过程

阅读数 3916

在本系列的上一篇文章中，我们学习了WMS的诞生，WMS被创建后，它的重要的成员有哪些？Window添加过程的... 博文 来自： 刘望舒的专栏

## Android输入法界面管理（打开/关闭/状态获取）

阅读数 1143

最近做一个带发表情的聊天界面，需要管理系统输入法的状态， 一、打开输入法窗口:[java] viewplain copyInputM... 博文 来自： blues的博客

## InputMethodManagerService 学习摘要

阅读数 2116

InputMethodManagerService学习摘要类的概述：接收屏幕开关、系统关闭对话框事件，启动、关闭输入法service... 博文 来自： 网事蒙尘

## Android：设置PopupWindow 的键盘弹出模式

阅读数 3445

设置弹出窗体需要软键盘，mPopupWindow.setInputMethodMode(Popupwindows.iNPUT_METHOD_NEEDED... 博文 来自： zhongyun_0602的...

## PopupWindow遮住虚拟键盘

阅读数 700

PopupWindow遮住虚拟导航键盘当手机有虚拟导航栏时，使用PopupWindow时，可能会导致弹出框，遮住虚拟导... 博文 来自： storm1314888的...

## 创建自定义输入法

阅读数 1606

注：参考资料：http://blog.csdn.net/dreamintheworld/article/details/50917804http://blog.csdn.net/dreamin... 博文 来自： nicolelili1的专栏

## 基于PyTorch的深度学习入门教程（六）——数据并行化

阅读数 4720

前言本文参考PyTorch官网的教程，分为五个基本模块来介绍PyTorch。为了避免文章过长，这五个模块分别在五篇... 博文 来自： 雁回晴空的博客专栏

## Android实现QQ分享及注意事项

阅读数 5547

一、获取APPID和帮助文档可以参看新手引导和接入说明：http://wiki.open.qq.com/wiki/移动应用接入wiki索引分... 博文 来自： 水寒

## CNN笔记：通俗理解卷积神经网络

阅读数 21万+

通俗理解卷积神经网络（cs231n与5月dl班课程笔记） 1 前言 2012年我在北京组织过8期machine l... 博文 来自： 结构之法 算法之道

## Android开发本地及网络Mp3音乐播放器(十二)创建NetMusicListAdapter、SearchResult显示...

阅读数 9291

实现功能： 实现NetMusicListAdapter（网络音乐列表适配器） 实现SearchResult（搜索音乐对象） 使用Jsoup组... 博文 来自： iwanghang(一个播...

## 【小程序】微信小程序开发实践

阅读数 25万+

帐号相关流程注册范围 企业 政府 媒体 其他组织换句话讲就是不让个人开发者注册。 :)填写企业信息不能使用和之前... 博文 来自： 小雨同学的技术博客

## HttpClient使用详解

阅读数 80万+

Http协议的重要性相信不用我多说了，HttpClient相比传统JDK自带的URLConnection，增加了易用性和灵活性（具... 博文 来自： 鹏霄万里展雄飞

## android客户端与服务器端交互 如何保持session

阅读数 4万+

最近在开发项目的过程中，遇到android与web服务器要在同一session下通信的问题。 在解决问题前先回顾下Sessi... 博文 来自： charming的专栏

## 三菱FX系列PLC与PC通讯的实现之专有协议（计算机联接）的程序设计之一

阅读数 1万+

阅读内容为：FX系列微型可编程控制器用户手册（通讯篇）中计算机链接功能章节。 采用本方法通信，pc端的实现... 博文 来自： pengjc2001的博客

## 如何在ArcGIS Online中构建自己的应用程序模板初级篇-显示地图

阅读数 4万+

开发ArcGIS Online应用程序模板之前，需要了解怎么使用ArcGIS API for JavaScript。 在ArcGIS Online当中如... 博文 来自： ArcGIS产品与技术...

## 再谈iOS 7的手势滑动返回功能

阅读数 8万+

之前随手写过一篇《使用UIScreenEdgePanGestureRecognizer实现swipe to pop效果》，挺粗糙的。现在使用默... 博文 来自： JasonLee的专栏

## [ASP.NET]二维码的创建

阅读数 5439

又好一段时间没有写写东西了，继续回归原来的模式，多做记录，最近要实现个unity的二维码方面的功能，首先就要... 博文 来自： 学无止境的专栏

## jquery/js实现一个网页同时调用多个倒计时(最新的)

阅读数 44万+

jquery/js实现一个网页同时调用多个倒计时(最新的) 最近需要网页添加多个倒计时. 查阅网络,基本上都是千遍一律的... 博文 来自： Websites

## 将Excel文件导入数据库（POI+Excel+MySQL+jsp页面导入）第一次优化

阅读数 3万+

本篇文章是根据我的上篇博客，给出的改进版，由于时间有限，仅做了一个简单的优化。相关文章：将excel导入数据... 博文 来自： Lynn_Blog

## webService学习（二）—— 调用自定义对象参数

阅读数 2万+

webService学习（二）—— 调用自定义对象参数 本文主要内容： 1、如何通过idea进行webService Client的简单... 博文 来自： 止水的专栏

## 人脸检测工具face_recognition的安装与应用

阅读数 5万+

人脸检测工具face_recognition的安装与应用 博文 来自： roguesir的博客

## C#实现开发windows服务实现自动从FTP服务器下载文件（自行设置分/时执行）

阅读数 2万+

最近在做一个每天定点从FTP自动下载节目.xml并更新到数据库的功能。首先想到用 FileSystemWatcher来监控下载... 博文 来自： kongwei521的专栏

## eclipse复制粘贴卡死

阅读数 2488

找了很多资料，最后总结在一起的解决eclipse复制粘贴时卡死的一些方案 博文 来自： 寒尘的专栏

## 微信支付V3微信公众号支付PHP教程(thinkPHP5公众号支付)/JSSDK的使用

阅读数 14万+

扫二维码关注，获取更多技术分享 本文承接之前发布的博客《 微信支付V3微信公众号支付PHP教程/thinkPHP5公众... 博文 来自： Marswill

## linux上安装Docker(非常简单的安装方法)

阅读数 20万+

最近比较有空，大四出来实习几个月了，作为实习狗的我，被叫去研究Docker了，汗汗！ Docker的三大核心概念：... 博文 来自： 我走小路的博客

## 如何在ubuntu 16.04上安装 RealSense（相机型号：Intel SR300）

阅读数 3633

前人栽树，后人乘凉~ 小白参考网上数篇教程（其实最主要是自己的安装记录，方便之后查找错误） https://github... 博文 来自： z17816876284的...

**强连通分量及缩点tarjan算法解析** 阅读数 57万+

强连通分量： 简言之 就是找环（每条边只走一次，两两可达） 孤立的一个点也是一个连通分量 使用tarjan算法 在... 博文 来自： 九野的博客

**【HTTP】Fiddler（一） - Fiddler简介** 阅读数 30万+

1.为什么是Fiddler? 抓包工具有很多，小到最常用的web调试工具firebug，达到通用的强大的抓包工具wireshark.为... 博文 来自： 专注、专心

**centos 查看命令源码** 阅读数 8万+

# yum install yum-utils 设置源: [base-src] name=CentOS-5.4 - Base src - baseurl=http://vault.ce... 博文 来自： linux/unix

**OpenCV+OpenGL 双目立体视觉三维重建** 阅读数 4万+

0.绪论这篇文章主要为了研究双目立体视觉的最终目标——三维重建，系统的介绍了三维重建的整体步骤。双目立体... 博文 来自： shiter编写程序的艺...

**mybatis一级缓存(session cache)引发的问题** 阅读数 2万+

mybatis一级缓存(session cache)引发的问题 博文 来自： flysharkym的专栏

**python图片处理类之~PIL.Image模块(ios android icon图标自动生成处理)** 阅读数 5万+

1.从pyCharm提示下载PIL包 http://www.pythonware.com/products/pil/ 2.解压后，进入到目录下 cd /Users/ji... 博文 来自： 专注于cocos+unit...

**Libusb库在Android下的使用例程** 阅读数 6807

转载请注明：http://blog.csdn.net/hubbybob1/article/details/54863662 阅读本文前清先了解相关基础内容，操... 博文 来自： hubbybob1专栏

**Hadoop+HBase完全分布式安装** 阅读数 4313

记录下完全分布式HBase数据库安装步骤准备3台机器：10.202.7.191 / 10.202.7.139 / 10.202.9.89所需准备的Jar包... 博文 来自： Dobbin

**DataTables 的 实例 《一》** 阅读数 1万+

1.加载需要的js/css文件 2. function del(id){ alert(id); } var table; $(document).ready(function(... 博文 来自： 辛修灿的博客

**SpringAOP拦截Controller,Service实现日志管理(自定义注解的方式)** 阅读数 12万+

首先我们为什么需要做日志管理，在现实的上线中我们经常会遇到系统出现异常或者问题。这个时候就马上打开CRT... 博文 来自： czmchen的专栏

**关于SpringBoot bean无法注入的问题（与文件包位置有关）** 阅读数 17万+

问题场景描述整个项目通过Maven构建，大致结构如下： 核心Spring框架一个module spring-boot-base service... 博文 来自： 开发随笔

**Android平台Camera实时滤镜实现方法探讨(五)--GLSurfaceView实现Camera预览** 阅读数 2万+

前面有一篇探讨了如何在片段着色器中将YUV数据转换为RGB数据并显示，但采用samplerExternalOES将SurfaceTe... 博文

**JavaWeb多文件上传及zip打包下载** 阅读数 6536

项目中经常会使用到文件上传及下载的功能。本篇文章总结场景在JavaWeb环境下，多文件上传及批量打包下载功能... 博文 来自： kidQ的博客

**R语言逻辑回归、ROC曲线和十折交叉验证** 阅读数 5万+

自己整理编写的逻辑回归模板，作为学习笔记记录分享。数据集用的是14个自变量Xi，一个因变量Y的australian数据... 博文 来自： Tiaaaaa的博客



罗升阳

博客专家

关注

| 原创 | 粉丝 | 喜欢 | 评论 |
| --- | --- | --- | --- |
| 188 | 3万+ | 390 | 8562 |

等级： 博客 8 访问： 1287万+

积分： 4万+ 排名： 110

勋章：

**进击的程序员**

《Android系统源代码情景分析》一书正在进击的程序员网(http://0xcc0xcd.com)中连载，点击进入

**最新文章**

Android WebView硬件加速渲染网页UI的过程分析

Android WebView执行GPU命令的过程分析

Android WebView启动Chromium渲染引擎的过程分析

Android WebView加载Chromium动态库的过程分析

22

20

**归档**

2017年1月 1篇

2016年12月 3篇

2016年11月 4篇

2016年10月 3篇

2016年9月 4篇

展开

**热门文章**

那两年炼就的Android内功修养
阅读数 450340

Android进程间通信（IPC）机制Binder简要介绍和学习计划
阅读数 419826

Android应用程序启动过程源代码分析
阅读数 404002

在Ubuntu上下载、编译和安装Android最新源代码
阅读数 270574

Android硬件抽象层（HAL）概要介绍和学习计划
阅读数 219890

**最新评论**

那两年炼就的Android内功修养
lsswshxcg：罗老师能给我发发你的Android学习视频吗

Android应用程序消息处理机制...
zhuhuang521：[reply]xiangyuxiangyu123[/reply] 情商为0

Android WebView硬件...
qq475703980：学习了

Android运行时ART简要介绍...
Johnny2017：罗总，不写博了，我们可以通过哪个途径追随你呢。

《Android系统源代码情景分析...
wdnwdn520：[reply]shi_xin[/reply] 老罗在oppo