Android InputMethod 源码分析，显示输入法流程

2017年05月06日 19:18:47　JieQiong1　阅读数：7620

1.简介

本文基于 android N，借鉴 http://blog.csdn.net/huangyabin001/article/details/28434989
，记录一下输入法显示的流程，相当于一篇读书笔记，方便记忆与学习

大体流程如下：

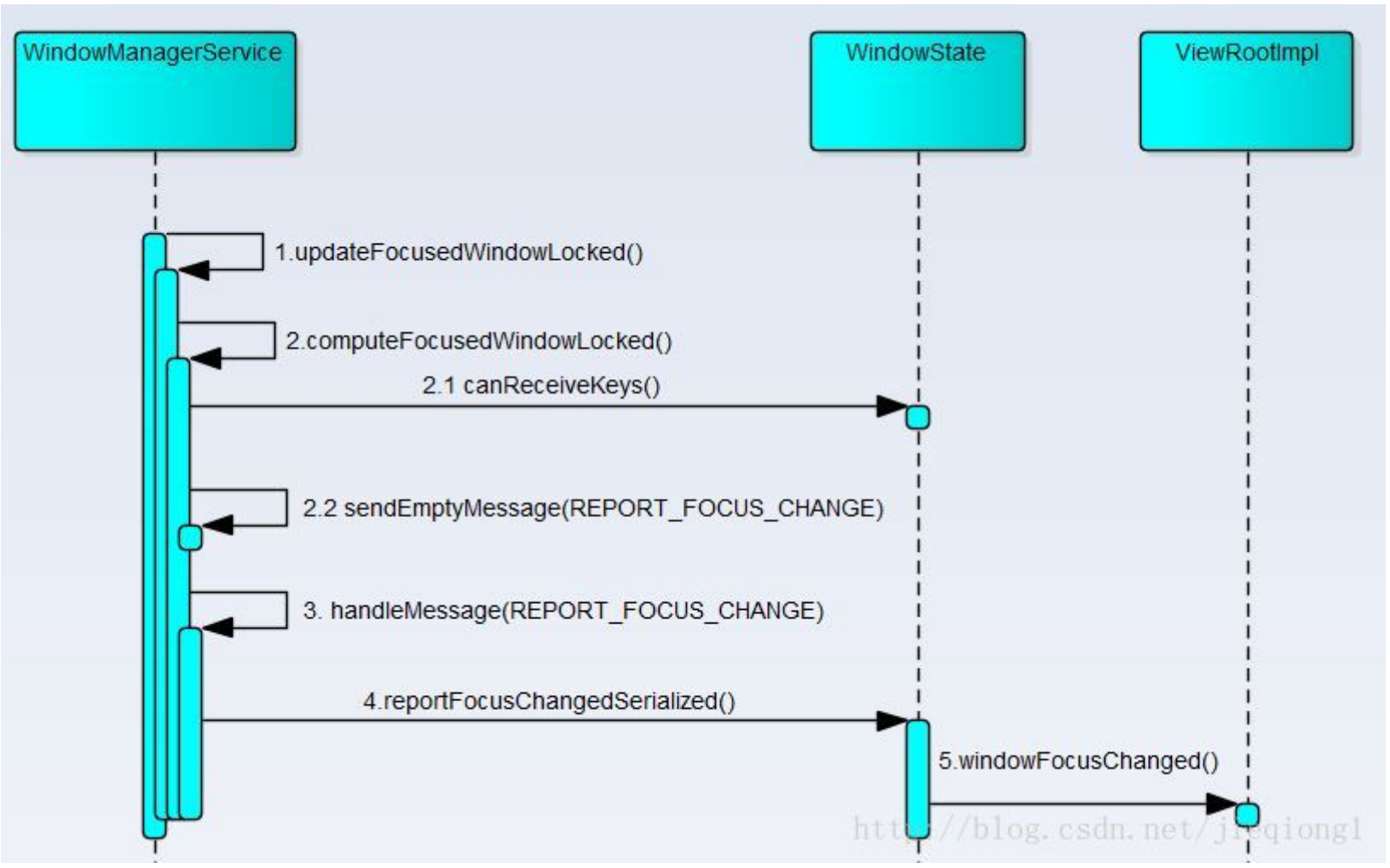
- InputMethodManagerService（下文也称IMMS）负责管理系统的所有输入法，包括输入法service(InputMethodService简称IMS)加载及切换。
- 程序获得焦点时，就会通过 InputMethodManager 向 InputMethodManagerService 发出请求绑定自己到当前输入法上。
- 当程序的某个需要输入法的view比如 EditView 获得焦点时，就会通过 InputMethodManager 向 InputMethodManagerService 请求显示输入法，而这时 InputMethodManagerService 收到请求后，会将请求的 EditText 的数据通信接口发送给当前输入法，并请求显示输入法。输入法收到请求后，就显示自己的 UI dialog,同时保存目标 view 的数据结构，当用户实现输入后，直接通过 view 的数据通信接口将字符传递到对应的 View 。接下来就来分析这些过程。

2. InputMethodManager 创建

```
// ViewRootImpl.java
public final class ViewRootImpl implements ViewParent,
        View.AttachInfo.Callbacks, ThreadedRenderer.HardwareDrawCallbacks {
    public ViewRootImpl(Context context, Display display) {
        ...
        mWindowSession = WindowManagerGlobal.getWindowSession();
        ...
    }

// WindowManagerGlobal.java
    public static IWindowSession getWindowSession() {
        synchronized (WindowManagerGlobal.class) {
            if (sWindowSession == null) {
                try {
                    // 生成 InputMethodManager 实例
                    InputMethodManager imm = InputMethodManager.getInstance();
                    IWindowManager windowManager = getWindowManagerService();
                    sWindowSession = windowManager.openSession(
                            new IWindowSessionCallback.Stub() {
                                @Override
                                public void onAnimatorScaleChanged(float scale) {
                                    ValueAnimator.setDurationScale(scale);
                                }
                            },
                            imm.getClient(), imm.getInputContext());
                } catch (RemoteException e) {
                    throw e.rethrowFromSystemServer();
                }
            }
            return sWindowSession;
        }
    }

// InputMethodManager.java
     /**
     * Retrieve the global InputMethodManager instance, creating it if it
     * doesn't already exist.
     * @hide
     */
     public static InputMethodManager getInstance() {
         synchronized (InputMethodManager.class) {

```

```
            if (sInstance == null) {
                IBinder b = ServiceManager.getService(Context.INPUT_METHOD_SERVICE);
                IInputMethodManager service = IInputMethodManager.Stub.asInterface(b);
                sInstance = new InputMethodManager(service, Looper.getMainLooper());
            }
            return sInstance;
        }
    }
```

3. window 获得焦点



流程图上传后不太清晰，在csdn中上传了原图：http://download.csdn.net/detail/jieqiong1/9835293

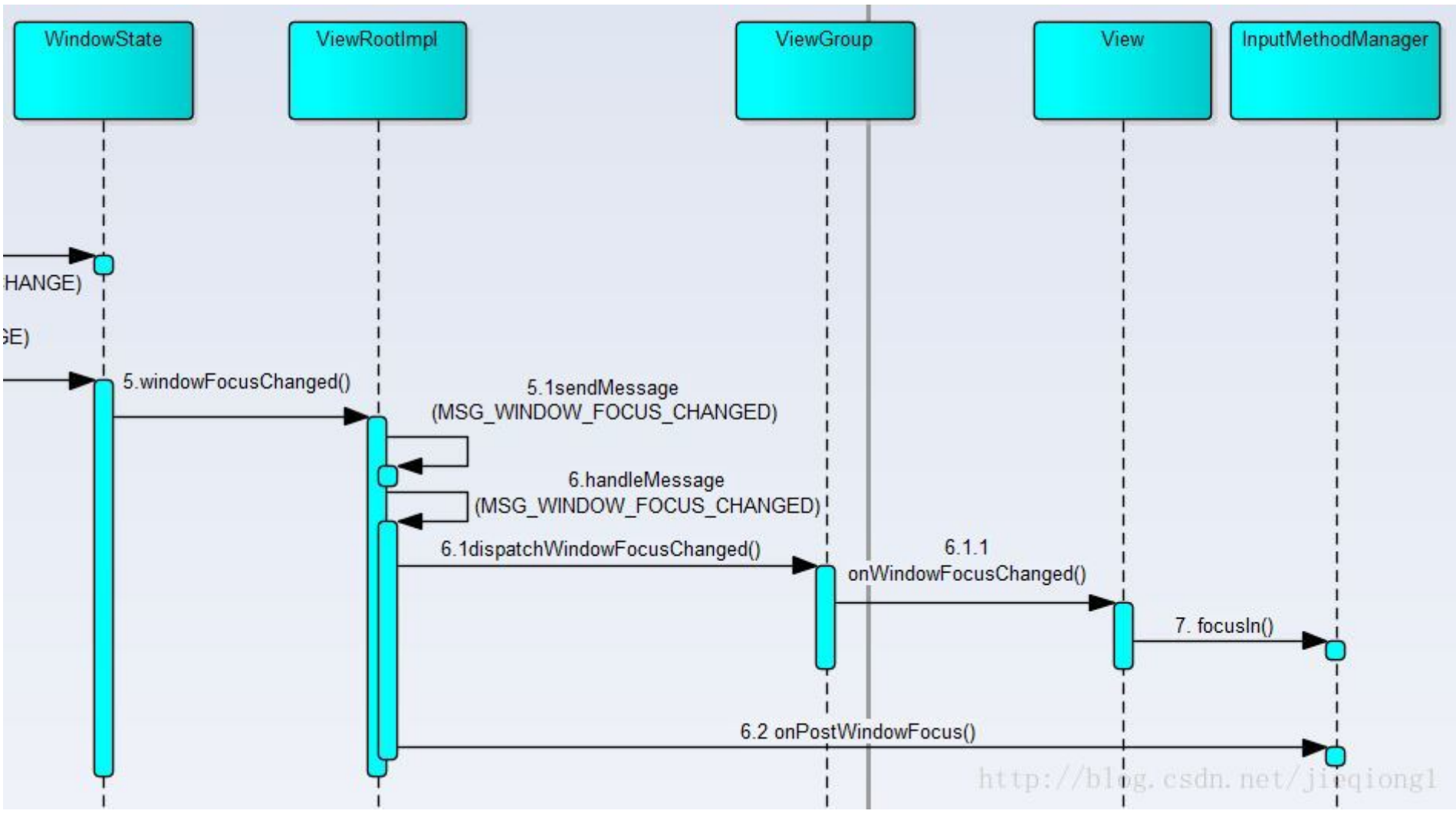
```
// WindowManagerService.java
    boolean updateFocusedWindowLocked(int mode, boolean updateInputWindows) {
        //计算焦点 window
        WindowState newFocus = computeFocusedWindowLocked();
        if (mCurrentFocus != newFocus) {
            Trace.traceBegin(Trace.TRACE_TAG_WINDOW_MANAGER, "wmUpdateFocus");
            // This check makes sure that we don't already have the focus
            // change message pending.
            mH.removeMessages(H.REPORT_FOCUS_CHANGE);
            mH.sendEmptyMessage(H.REPORT_FOCUS_CHANGE);
            ...
            return true;
        }
        return false;
    }

    private WindowState computeFocusedWindowLocked() {
        final int displayCount = mDisplayContents.size();
        for (int i = 0; i < displayCount; i++) {
            final DisplayContent displayContent = mDisplayContents.valueAt(i);
            WindowState win = findFocusedWindowLocked(displayContent);
            if (win != null) {
                return win;
            }
        }
        return null;
    }
```

```
    }

    // 找出 top 需要获得焦点的 window
    WindowState findFocusedWindowLocked(DisplayContent displayContent) {
        final WindowList windows = displayContent.getWindowList();
        for (int i = windows.size() - 1; i >= 0; i--) {
            final WindowState win = windows.get(i);

            if (localLOGV || DEBUG_FOCUS) Slog.v(
                TAG_WM, "Looking for focus: " + i
                + " = " + win
                + ", flags=" + win.mAttrs.flags
                + ", canReceive=" + win.canReceiveKeys());

            // 判断 window 是否可以获取焦点
            if (!win.canReceiveKeys()) {
                continue;
            }

            // win.mAppToken != null win描述的是一个Activity窗口
            AppWindowToken wtoken = win.mAppToken;

            // If this window's application has been removed, just skip it.
            if (wtoken != null && (wtoken.removed || wtoken.sendingToBottom)) {
                if (DEBUG_FOCUS) Slog.v(TAG_WM, "Skipping " + wtoken + " because "
                        + (wtoken.removed ? "removed" : "sendingToBottom"));
                continue;
            }

            // Descend through all of the app tokens and find the first that either matches
            // win.mAppToken (return win) or mFocusedApp (return null).
            // mFocusedApp 是 top Activity , 下边的逻辑是为了确保焦点window的app 必须是焦点程序上的 , 主要是为了检测出错误
            if (wtoken != null && win.mAttrs.type != TYPE_APPLICATION_STARTING &&
                    mFocusedApp != null) {
                ArrayList<Task> tasks = displayContent.getTasks();
                for (int taskNdx = tasks.size() - 1; taskNdx >= 0; --taskNdx) {
                    AppTokenList tokens = tasks.get(taskNdx).mAppTokens;
                    int tokenNdx = tokens.size() - 1;
                    for ( ; tokenNdx >= 0; --tokenNdx) {
                        final AppWindowToken token = tokens.get(tokenNdx);
                        if (wtoken == token) {
                            break;
                        }
                        if (mFocusedApp == token && token.windowsAreFocusable()) {
                            // Whoops, we are below the focused app whose windows are focusable...
                            // No focus for you!!!
                            if (localLOGV || DEBUG_FOCUS_LIGHT) Slog.v(TAG_WM,
                                    "findFocusedWindow: Reached focused app=" + mFocusedApp);
                            return null;
                        }
                    }
                    if (tokenNdx >= 0) {
                        // Early exit from loop, must have found the matching token.
                        break;
                    }
                }
            }

            if (DEBUG_FOCUS_LIGHT) Slog.v(TAG_WM, "findFocusedWindow: Found new focus @ " + i +
                    " = " + win);
            return win;
        }

        if (DEBUG_FOCUS_LIGHT) Slog.v(TAG_WM, "findFocusedWindow: No focusable windows.");
        return null;
    }
```

```
@Override
public void handleMessage(Message msg) {
    if (DEBUG_WINDOW_TRACE) {
        Slog.v(TAG_WM, "handleMessage: entry what=" + msg.what);
    }
    switch (msg.what) {
        case REPORT_FOCUS_CHANGE: {
            WindowState lastFocus;
            WindowState newFocus;

            AccessibilityController accessibilityController = null;

            synchronized(mWindowMap) {
                // TODO(multidisplay): Accessibility supported only of default desiplay.
                if (mAccessibilityController != null && getDefaultDisplayContentLocked()
                        .getDisplayId() == Display.DEFAULT_DISPLAY) {
                    accessibilityController = mAccessibilityController;
                }

                lastFocus = mLastFocus;
                newFocus = mCurrentFocus;
                if (lastFocus == newFocus) {
                    // Focus is not changing, so nothing to do.
                    return;
                }
                mLastFocus = newFocus;
                if (DEBUG_FOCUS_LIGHT) Slog.i(TAG_WM, "Focus moving from " + lastFocus +
                        " to " + newFocus);
                if (newFocus != null && lastFocus != null
                        && !newFocus.isDisplayedLw()) {
                    //Slog.i(TAG_WM, "Delaying loss of focus...");
                    mLosingFocus.add(lastFocus);
                    lastFocus = null;
                }
            }

            // First notify the accessibility manager for the change so it has
            // the windows before the newly focused one starts firing eventgs.
            if (accessibilityController != null) {
                accessibilityController.onWindowFocusChangedNotLocked();
            }

            //System.out.println("Changing focus from " + lastFocus
            //                   + " to " + newFocus);
            if (newFocus != null) {
                if (DEBUG_FOCUS_LIGHT) Slog.i(TAG_WM, "Gaining focus: " + newFocus);
                // 通知新的焦点程序 获得了焦点
                newFocus.reportFocusChangedSerialized(true, mInTouchMode);
                notifyFocusChanged();
            }

            if (lastFocus != null) {
                if (DEBUG_FOCUS_LIGHT) Slog.i(TAG_WM, "Losing focus: " + lastFocus);
                // 通知老的焦点程序 获得了焦点
                lastFocus.reportFocusChangedSerialized(false, mInTouchMode);
            }
        } break;
        ...
    }


// WindowState.java
    /**
```

```
    /**
     * Report a focus change.  Must be called with no locks held, and consistently
     * from the same serialized thread (such as dispatched from a handler).
     */
    public void reportFocusChangedSerialized(boolean focused, boolean inTouchMode) {
        try {
            //通过 Binder 告知 client端 其获得或失去了焦点
            mClient.windowFocusChanged(focused, inTouchMode);
        } catch (RemoteException e) {
        }
        if (mFocusCallbacks != null) {
            final int N = mFocusCallbacks.beginBroadcast();
            for (int i=0; i<N; i++) {
                IWindowFocusObserver obs = mFocusCallbacks.getBroadcastItem(i);
                try {
                    if (focused) {
                        obs.focusGained(mWindowId.asBinder());
                    } else {
                        obs.focusLost(mWindowId.asBinder());
                    }
                } catch (RemoteException e) {
                }
            }
            mFocusCallbacks.finishBroadcast();
        }
    }
```

4.程序变更焦点，程序获得焦点变更事件

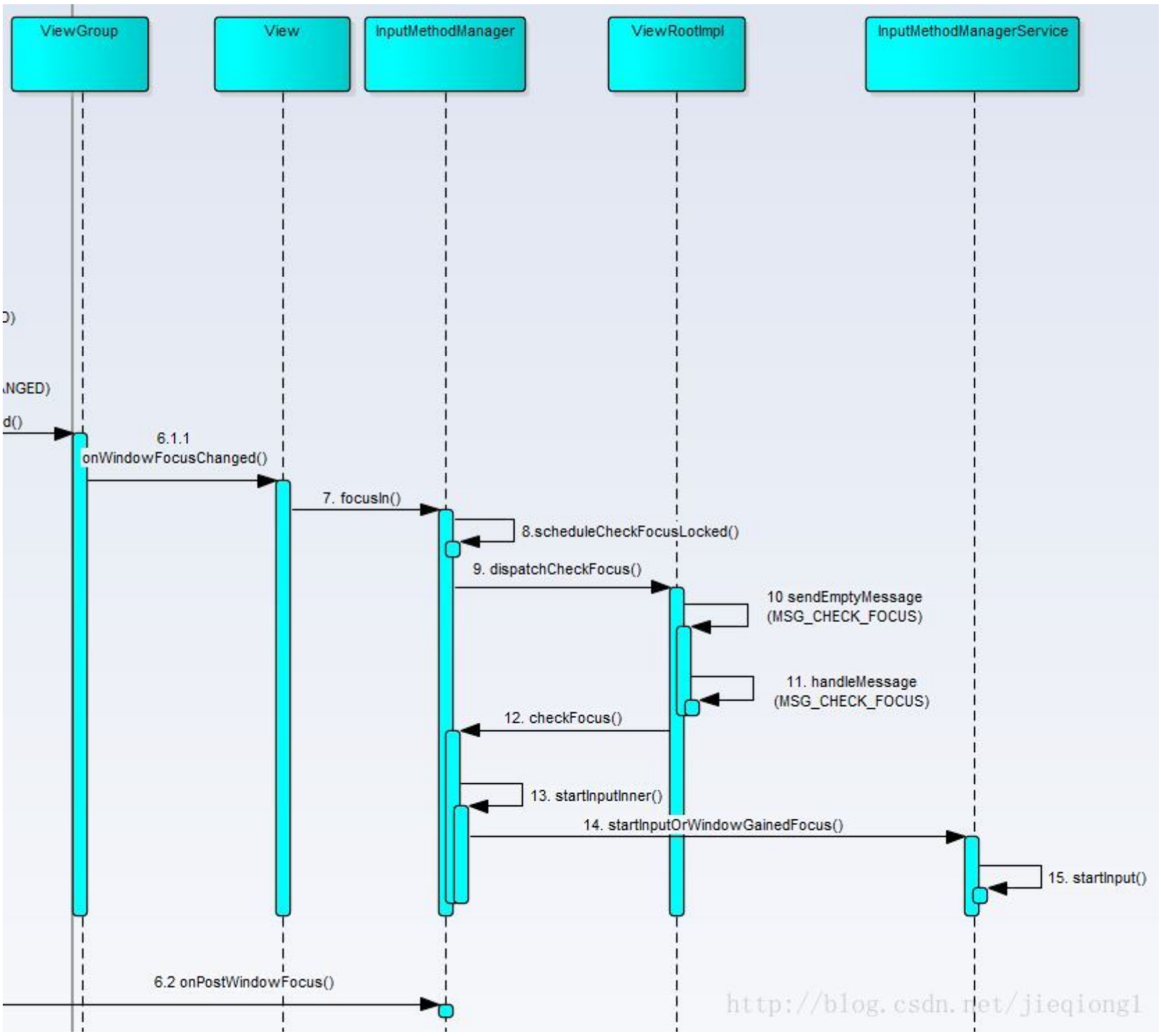



```
// ViewRootImpl.java
    public void windowFocusChanged(boolean hasFocus, boolean inTouchMode) {
        Message msg = Message.obtain();
        msg.what = MSG_WINDOW_FOCUS_CHANGED;
        msg.arg1 = hasFocus ? 1 : 0;
        msg.arg2 = inTouchMode ? 1 : 0;
        mHandler.sendMessage(msg);
    }

    @Override
```

```
12      public void handleMessage(Message msg) {
13          switch (msg.what) {
14              case MSG_WINDOW_FOCUS_CHANGED: {
15                  if (mAdded) {
16                      boolean hasWindowFocus = msg.arg1 != 0;
17                      mAttachInfo.mHasWindowFocus = hasWindowFocus;
18
19                      profileRendering(hasWindowFocus);
20
21                      if (hasWindowFocus) {
22                          ...
23                      }
24
25                      mLastWasImTarget = WindowManager.LayoutParams
26                              .mayUseInputMethod(mWindowAttributes.flags);
27
28                      InputMethodManager imm = InputMethodManager.peekInstance();
29                      if (imm != null && mLastWasImTarget && !isInLocalFocusMode()) {
30                          imm.onPreWindowFocus(mView, hasWindowFocus);
31                      }
32                      if (mView != null) {
33                          mAttachInfo.mKeyDispatchState.reset();
34                          // 6.1 调用根 view的 dispatchWindowFocusChanged() , 通知view程序获得焦点
35                          mView.dispatchWindowFocusChanged(hasWindowFocus);
36                          mAttachInfo.mTreeObserver.dispatchOnWindowFocusChange(hasWindowFocus);
37                      }
38
39                      // Note: must be done after the focus change callbacks,
40                      // so all of the view state is set up correctly.
41                      if (hasWindowFocus) {
42                          if (imm != null && mLastWasImTarget && !isInLocalFocusMode()) {
43                              // 6.2 通知 InputMethodManager 该 window 获得焦点
44                              imm.onPostWindowFocus(mView, mView.findFocus(),
45                                      mWindowAttributes.softInputMode,
46                                      !mHasHadWindowFocus, mWindowAttributes.flags);
47                          }
48                          // Clear the forward bit.  We can just do this directly, since
49                          // the window manager doesn't care about it.
50                          mWindowAttributes.softInputMode &=
51                                  ~WindowManager.LayoutParams.SOFT_INPUT_IS_FORWARD_NAVIGATION;
52                          ((WindowManager.LayoutParams)mView.getLayoutParams())
53                                  .softInputMode &=
54                                      ~WindowManager.LayoutParams.SOFT_INPUT_IS_FORWARD_NAVIGATION;
55                          mHasHadWindowFocus = true;
56                      }
57                  }
58              } break;
59
60              ...
61              }
62      }
```

一.1 焦点View向IMMS请求绑定输入法

6.1 之后的流程




```
// ViewGroup.java
    @Override
    public void dispatchWindowFocusChanged(boolean hasFocus) {
        super.dispatchWindowFocusChanged(hasFocus);
        final int count = mChildrenCount;
        final View[] children = mChildren;
        for (int i = 0; i < count; i++) {
            children[i].dispatchWindowFocusChanged(hasFocus);
        }
    }
// View.java
    /**
     * Called when the window containing this view gains or loses window focus.
     * ViewGroups should override to route to their children.
     *
     * @param hasFocus True if the window containing this view now has focus,
     *        false otherwise.
     */
    public void dispatchWindowFocusChanged(boolean hasFocus) {
        onWindowFocusChanged(hasFocus);
    }

    /**
     * Called when the window containing this view gains or loses focus.  Note
     * that this is separate from view focus: to receive key events, both
     * your view and its window must have focus.  If a window is displayed
     * on top of yours that takes input focus, then your own window will lose
     * focus but the view focus will remain unchanged.
     *
```

```
     * @param hasWindowFocus True if the window containing this view now has
     *        focus, false otherwise.
     */
      public void onWindowFocusChanged(boolean hasWindowFocus) {
          InputMethodManager imm = InputMethodManager.peekInstance();
          if (!hasWindowFocus) {
              if (isPressed()) {
                  setPressed(false);
              }
              if (imm != null && (mPrivateFlags & PFLAG_FOCUSED) != 0) {
                  imm.focusOut(this);
              }
              removeLongPressCallback();
              removeTapCallback();
              onFocusLost();
          } else if (imm != null && (mPrivateFlags & PFLAG_FOCUSED) != 0) {
              // 获得焦点的 view 通过 InputMethodManager 向 Service 通知自己获得焦点
              imm.focusIn(this);
          }
          refreshDrawableState();
      }


// InputMethodManager.java
      /**
     * Call this when a view receives focus.
     * @hide
     */
      public void focusIn(View view) {
          synchronized (mH) {
              focusInLocked(view);
          }
      }


// InputMethodManager.java
      /**
     * Call this when a view receives focus.
     * @hide
     */
      public void focusIn(View view) {
          synchronized (mH) {
              focusInLocked(view);
          }
      }

      void focusInLocked(View view) {
          if (DEBUG) Log.v(TAG, "focusIn: " + dumpViewInfo(view));

          if (view != null && view.isTemporarilyDetached()) {
              // This is a request from a view that is temporarily detached from a window.
              if (DEBUG) Log.v(TAG, "Temporarily detached view, ignoring");
              return;
          }

          if (mCurRootView != view.getRootView()) {
              // This is a request from a window that isn't in the window with
              // IME focus, so ignore it.
              if (DEBUG) Log.v(TAG, "Not IME target window, ignoring");
              return;
          }

          mNextServedView = view;//保存焦点view的变量
          scheduleCheckFocusLocked(view);
      }

      static void scheduleCheckFocusLocked(View view) {
```

```
        ViewRootImpl viewRootImpl = view.getViewRootImpl();
        if (viewRootImpl != null) {
            viewRootImpl.dispatchCheckFocus();
        }
    }

// ViewRootImpl.java
    public void dispatchCheckFocus() {
        if (!mHandler.hasMessages(MSG_CHECK_FOCUS)) {
            // This will result in a call to checkFocus() below.
            mHandler.sendEmptyMessage(MSG_CHECK_FOCUS);
        }
    }

    case MSG_CHECK_FOCUS: {
        InputMethodManager imm = InputMethodManager.peekInstance();
        if (imm != null) {
            imm.checkFocus();
        }
    } break;

// InputMethodManager.java
     /**
    * @hide
    */
     public void checkFocus() {
          // 确认当前 focused view 是否已经调用过 startInputInner() 来绑定输入法，
          // 因为前面 mView.dispatchWindowFocusChanged() 已经完成了 focused view 的绑定，
          // 大部分情况下，该函数返回 false，不会再次调用 startInputInner()
         if (checkFocusNoStartInput(false)) {
             startInputInner(InputMethodClient.START_INPUT_REASON_CHECK_FOCUS, null, 0, 0, 0);
         }
     }

     private boolean checkFocusNoStartInput(boolean forceNewFocus) {
         // This is called a lot, so short-circuit before locking.
         if (mServedView == mNextServedView && !forceNewFocus) {
             return false;
         }

         final ControlledInputConnectionWrapper ic;
         synchronized (mH) {
             if (mServedView == mNextServedView && !forceNewFocus) {
                 return false;
             }
             if (DEBUG) Log.v(TAG, "checkFocus: view=" + mServedView
                     + " next=" + mNextServedView
                     + " forceNewFocus=" + forceNewFocus
                     + " package="
                     + (mServedView != null ? mServedView.getContext().getPackageName() : "<none>"));

             if (mNextServedView == null) {
                 finishInputLocked();
                 // In this case, we used to have a focused view on the window,
                 // but no longer do.  We should make sure the input method is
                 // no longer shown, since it serves no purpose.
                 closeCurrentInput();
                 return false;
             }

             ic = mServedInputConnectionWrapper;

             mServedView = mNextServedView;
             mCurrentTextBoxAttribute = null;
             mCompletions = null;
```

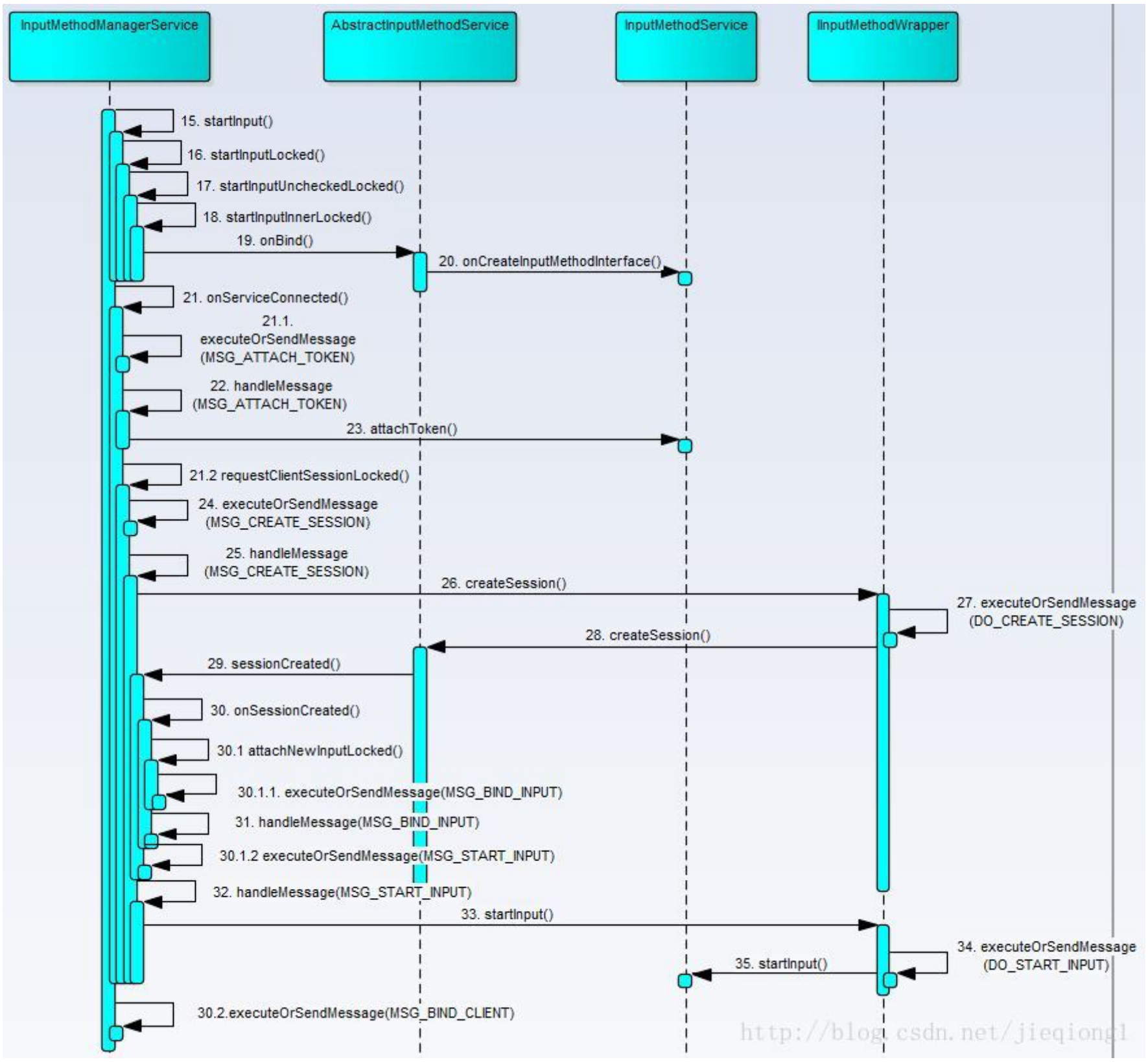
```
            mServedConnecting = true;
        }

        if (ic != null) {
            ic.finishComposingText();
        }

        return true;
    }

    boolean startInputInner(@InputMethodClient.StartInputReason final int startInputReason,
            IBinder windowGainingFocus, int controlFlags, int softInputMode,
            int windowFlags) {
        final View view;
        synchronized (mH) {
            // 获得上面的焦点view
            view = mServedView;

            // Make sure we have a window token for the served view.
            if (DEBUG) {
                Log.v(TAG, "Starting input: view=" + dumpViewInfo(view) +
                        " reason=" + InputMethodClient.getStartInputReason(startInputReason));
            }
            if (view == null) {
                if (DEBUG) Log.v(TAG, "ABORT input: no served view!");
                return false;
            }
        }

        // Now we need to get an input connection from the served view.
        // This is complicated in a couple ways: we can't be holding our lock
        // when calling out to the view, and we need to make sure we call into
        // the view on the same thread that is driving its view hierarchy.
        Handler vh = view.getHandler();
        if (vh == null) {
            // If the view doesn't have a handler, something has changed out
            // from under us, so just close the current input.
            // If we don't close the current input, the current input method can remain on the
            // screen without a connection.
            if (DEBUG) Log.v(TAG, "ABORT input: no handler for view! Close current input.");
            closeCurrentInput();
            return false;
        }
        if (vh.getLooper() != Looper.myLooper()) {
            // The view is running on a different thread than our own, so
            // we need to reschedule our work for over there.
            if (DEBUG) Log.v(TAG, "Starting input: reschedule to view thread");
            vh.post(new Runnable() {
                @Override
                public void run() {
                    startInputInner(startInputReason, null, 0, 0, 0);
                }
            });
            return false;
        }

        // Okay we are now ready to call into the served view and have it
        // do its stuff.
        // Life is good: let's hook everything up!
        EditorInfo tba = new EditorInfo();
        // Note: Use Context#getOpPackageName() rather than Context#getPackageName() so that the
        // system can verify the consistency between the uid of this process and package name passed
        // from here. See comment of Context#getOpPackageName() for details.
        tba.packageName = view.getContext().getOpPackageName();
        tba.fieldId = view.getId();
        // 创建数据通信连接接口 InputConnection
```

```
        // InputMethodService 后面就是通过这个connection将输入法的字符传给该view
        InputConnection ic = view.onCreateInputConnection(tba);
        if (DEBUG) Log.v(TAG, "Starting input: tba=" + tba + " ic=" + ic);

        synchronized (mH) {
            // Now that we are locked again, validate that our state hasn't
            // changed.
            if (mServedView != view || !mServedConnecting) {
                // Something else happened, so abort.
                if (DEBUG) Log.v(TAG,
                        "Starting input: finished by someone else. view=" + dumpViewInfo(view)
                        + " mServedView=" + dumpViewInfo(mServedView)
                        + " mServedConnecting=" + mServedConnecting);
                return false;
            }

            // If we already have a text box, then this view is already
            // connected so we want to restart it.
            if (mCurrentTextBoxAttribute == null) {
                controlFlags |= CONTROL_START_INITIAL;
            }

            // Hook 'em up and let 'er rip.
            mCurrentTextBoxAttribute = tba;
            mServedConnecting = false;
            if (mServedInputConnectionWrapper != null) {
                mServedInputConnectionWrapper.deactivate();
                mServedInputConnectionWrapper = null;
            }
            ControlledInputConnectionWrapper servedContext;
            final int missingMethodFlags;
            if (ic != null) {
                mCursorSelStart = tba.initialSelStart;
                mCursorSelEnd = tba.initialSelEnd;
                mCursorCandStart = -1;
                mCursorCandEnd = -1;
                mCursorRect.setEmpty();
                mCursorAnchorInfo = null;
                final Handler icHandler;
                missingMethodFlags = InputConnectionInspector.getMissingMethodFlags(ic);
                if ((missingMethodFlags & InputConnectionInspector.MissingMethodFlags.GET_HANDLER)
                        != 0) {
                    // InputConnection#getHandler() is not implemented.
                    icHandler = null;
                } else {
                    icHandler = ic.getHandler();
                }
                // 将 InputConnection 封装为 binder 对象，这个是真正可以实现跨进程通信的封装类
                servedContext = new ControlledInputConnectionWrapper(
                        icHandler != null ? icHandler.getLooper() : vh.getLooper(), ic, this);
            } else {
                servedContext = null;
                missingMethodFlags = 0;
            }
            mServedInputConnectionWrapper = servedContext;

            try {
                if (DEBUG) Log.v(TAG, "START INPUT: view=" + dumpViewInfo(view) + " ic="
                        + ic + " tba=" + tba + " controlFlags=#"
                        + Integer.toHexString(controlFlags));
                final InputBindResult res = mService.startInputOrWindowGainedFocus(
                        startInputReason, mClient, windowGainingFocus, controlFlags, softInputMode,
                        windowFlags, tba, servedContext, missingMethodFlags);
                if (DEBUG) Log.v(TAG, "Starting input: Bind result=" + res);
                if (res != null) {
                    if (res.id != null) {
```

```
                setInputChannelLocked(res.channel);
                mBindSequence = res.sequence;
                // 获得输入法的通信接口
                mCurMethod = res.method;
                mCurId = res.id;
                mNextUserActionNotificationSequenceNumber =
                        res.userActionNotificationSequenceNumber;
                if (mServedInputConnectionWrapper != null) {
                    mServedInputConnectionWrapper.setInputMethodId(mCurId);
                }
            } else {
                if (res.channel != null && res.channel != mCurChannel) {
                    res.channel.dispose();
                }
                if (mCurMethod == null) {
                    // This means there is no input method available.
                    if (DEBUG) Log.v(TAG, "ABORT input: no input method!");
                    return true;
                }
            }
        } else {
            if (startInputReason
                    == InputMethodClient.START_INPUT_REASON_WINDOW_FOCUS_GAIN) {
                // We are here probably because of an obsolete window-focus-in message sent
                // to windowGainingFocus.  Since IMMS determines whether a Window can have
                // IME focus or not by using the latest window focus state maintained in the
                // WMS, this kind of race condition cannot be avoided.  One obvious example
                // would be that we have already received a window-focus-out message but the
                // UI thread is still handling previous window-focus-in message here.
                // TODO: InputBindResult should have the error code.
                if (DEBUG) Log.w(TAG, "startInputOrWindowGainedFocus failed. "
                        + "Window focus may have already been lost. "
                        + "win=" + windowGainingFocus + " view=" + dumpViewInfo(view));
                if (!mActive) {
                    // mHasBeenInactive is a latch switch to forcefully refresh IME focus
                    // state when an inactive (mActive == false) client is gaining window
                    // focus. In case we have unnecessary disable the latch due to this
                    // spurious wakeup, we re-enable the latch here.
                    // TODO: Come up with more robust solution.
                    mHasBeenInactive = true;
                }
            }
        }
        if (mCurMethod != null && mCompletions != null) {
            try {
                mCurMethod.displayCompletions(mCompletions);
            } catch (RemoteException e) {
            }
        }
    } catch (RemoteException e) {
        Log.w(TAG, "IME died: " + mCurId, e);
    }
}

return true;
}

// InputMethodManagerService.java
@Override
public InputBindResult startInputOrWindowGainedFocus(
        /* @InputMethodClient.StartInputReason */ final int startInputReason,
        IInputMethodClient client, IBinder windowToken, int controlFlags, int softInputMode,
        int windowFlags, @Nullable EditorInfo attribute, IInputContext inputContext,
        /* @InputConnectionInspector.missingMethods */ final int missingMethods) {
    if (windowToken != null) {
        // focusIn 不走该分支
```

```
            return windowGainedFocus(startInputReason, client, windowToken, controlFlags,
                    softInputMode, windowFlags, attribute, inputContext, missingMethods);
        } else {
            // view 获得焦点，IMMS将这个 view 和 输入法绑定
            return startInput(startInputReason, client, inputContext, missingMethods, attribute,
                    controlFlags);
        }
    }
```

一.2 IMMS处理view绑定输入法事件




```
1  // InputMethodManagerService.java
2      @Override
3      public InputBindResult startInputOrWindowGainedFocus(
4              /* @InputMethodClient.StartInputReason */ final int startInputReason,
5              IInputMethodClient client, IBinder windowToken, int controlFlags, int softInputMode,
6              int windowFlags, @Nullable EditorInfo attribute, IInputContext inputContext,
7              /* @InputConnectionInspector.missingMethods */ final int missingMethods) {
8          if (windowToken != null) {
9              // focusIn 不走该分支
10             return windowGainedFocus(startInputReason, client, windowToken, controlFlags,
11                     softInputMode, windowFlags, attribute, inputContext, missingMethods);
12         } else {
13             // view 获得焦点，IMMS将这个 view 和 输入法绑定
14             return startInput(startInputReason, client, inputContext, missingMethods, attribute,
15                     controlFlags);
16         }
```

```
        }
    }

    private InputBindResult startInput(
            /* @InputMethodClient.StartInputReason */ final int startInputReason,
            IInputMethodClient client, IInputContext inputContext,
            /* @InputConnectionInspector.missingMethods */ final int missingMethods,
            @Nullable EditorInfo attribute, int controlFlags) {
        if (!calledFromValidUser()) {
            return null;
        }
        synchronized (mMethodMap) {
            if (DEBUG) {
                Slog.v(TAG, "startInput: reason="
                        + InputMethodClient.getStartInputReason(startInputReason)
                        + " client = " + client.asBinder()
                        + " inputContext=" + inputContext
                        + " missingMethods="
                        + InputConnectionInspector.getMissingMethodFlagsAsString(missingMethods)
                        + " attribute=" + attribute
                        + " controlFlags=#" + Integer.toHexString(controlFlags));
            }
            final long ident = Binder.clearCallingIdentity();
            try {
                return startInputLocked(startInputReason, client, inputContext, missingMethods,
                        attribute, controlFlags);
            } finally {
                Binder.restoreCallingIdentity(ident);
            }
        }
    }


    InputBindResult startInputLocked(
            /* @InputMethodClient.StartInputReason */ final int startInputReason,
            IInputMethodClient client, IInputContext inputContext,
            /* @InputConnectionInspector.missingMethods */ final int missingMethods,
            @Nullable EditorInfo attribute, int controlFlags) {
        // If no method is currently selected, do nothing.
        if (mCurMethodId == null) {
            return mNoBinding;
        }

        // 程序在 Service 端 对应的数据结构
        ClientState cs = mClients.get(client.asBinder());
        ...
        return startInputUncheckedLocked(cs, inputContext, missingMethods, attribute,
                controlFlags);
    }


    InputBindResult startInputUncheckedLocked(@NonNull ClientState cs, IInputContext inputContext,
            /* @InputConnectionInspector.missingMethods */ final int missingMethods,
            @NonNull EditorInfo attribute, int controlFlags) {
        // If no method is currently selected, do nothing.
        if (mCurMethodId == null) {
            return mNoBinding;
        }

        if (!InputMethodUtils.checkIfPackageBelongsToUid(mAppOpsManager, cs.uid,
                attribute.packageName)) {
            Slog.e(TAG, "Rejecting this client as it reported an invalid package name."
                    + " uid=" + cs.uid + " package=" + attribute.packageName);
            return mNoBinding;
        }

        if (mCurClient != cs) {
            // 如果新程序和当前活动的程序不同，那当前活动程序会输入法的绑定
```

```
            // 如果新程序和当前活动的程序不同，取消当前活动程序与输入法的绑定
            // Was the keyguard locked when switching over to the new client?
            mCurClientInKeyguard = isKeyguardLocked();
            // If the client is changing, we need to switch over to the new
            // one.
            unbindCurrentClientLocked(InputMethodClient.UNBIND_REASON_SWITCH_CLIENT);
            if (DEBUG) Slog.v(TAG, "switching to client: client="
                    + cs.client.asBinder() + " keyguard=" + mCurClientInKeyguard);

            // If the screen is on, inform the new client it is active
            if (mIsInteractive) {
                executeOrSendMessage(cs.client, mCaller.obtainMessageIO(
                        MSG_SET_ACTIVE, mIsInteractive ? 1 : 0, cs));
            }
        }

        // Bump up the sequence for this client and attach it.
        mCurSeq++;
        if (mCurSeq <= 0) mCurSeq = 1;
        // 将新程序设置为当前活动的程序
        mCurClient = cs;
        mCurInputContext = inputContext;
        mCurInputContextMissingMethods = missingMethods;
        mCurAttribute = attribute;

        // Check if the input method is changing.
        if (mCurId != null && mCurId.equals(mCurMethodId)) {
            if (cs.curSession != null) {
                // Fast case: if we are already connected to the input method,
                // then just return it.
                // 连接已经建立，开始绑定
                return attachNewInputLocked(
                        (controlFlags&InputMethodManager.CONTROL_START_INITIAL) != 0);
            }
            if (mHaveConnection) {
                if (mCurMethod != null) {
                    // 如果 输入法的连接 已经创建，直接传递给程序 client 端
                    // Return to client, and we will get back with it when
                    // we have had a session made for it.
                    requestClientSessionLocked(cs);
                    return new InputBindResult(null, null, mCurId, mCurSeq,
                            mCurUserActionNotificationSequenceNumber);
                } else if (SystemClock.uptimeMillis()
                        < (mLastBindTime+TIME_TO_RECONNECT)) {
                    // In this case we have connected to the service, but
                    // don't yet have its interface.  If it hasn't been too
                    // long since we did the connection, we'll return to
                    // the client and wait to get the service interface so
                    // we can report back.  If it has been too long, we want
                    // to fall through so we can try a disconnect/reconnect
                    // to see if we can get back in touch with the service.
                    return new InputBindResult(null, null, mCurId, mCurSeq,
                            mCurUserActionNotificationSequenceNumber);
                } else {
                    EventLog.writeEvent(EventLogTags.IMF_FORCE_RECONNECT_IME,
                            mCurMethodId, SystemClock.uptimeMillis()-mLastBindTime, 0);
                }
            }
        }
        // 启动输入法并建立连接
        return startInputInnerLocked();
    }

    InputBindResult startInputInnerLocked() {
        if (mCurMethodId == null) {
            return mNoBinding;

        }
```

```
        }

        if (!mSystemReady) {
            // If the system is not yet ready, we shouldn't be running third
            // party code.
            return new InputBindResult(null, null, mCurMethodId, mCurSeq,
                    mCurUserActionNotificationSequenceNumber);
        }

        InputMethodInfo info = mMethodMap.get(mCurMethodId);
        if (info == null) {
            throw new IllegalArgumentException("Unknown id: " + mCurMethodId);
        }

        unbindCurrentMethodLocked(true);
        //启动输入法Service
        mCurIntent = new Intent(InputMethod.SERVICE_INTERFACE);
        mCurIntent.setComponent(info.getComponent());
        mCurIntent.putExtra(Intent.EXTRA_CLIENT_LABEL,
                com.android.internal.R.string.input_method_binding_label);
        mCurIntent.putExtra(Intent.EXTRA_CLIENT_INTENT, PendingIntent.getActivity(
                mContext, 0, new Intent(Settings.ACTION_INPUT_METHOD_SETTINGS), 0));
        if (bindCurrentInputMethodService(mCurIntent, this, Context.BIND_AUTO_CREATE
                | Context.BIND_NOT_VISIBLE | Context.BIND_NOT_FOREGROUND
                | Context.BIND_SHOWING_UI)) {
            mLastBindTime = SystemClock.uptimeMillis();
            mHaveConnection = true;
            mCurId = info.getId();
            // mCurToken 是给输入法Service 来绑定输入法window的
            // 通过 mCurToken ，InputMethodManagerService 直接管理 输入法window
            mCurToken = new Binder();
            try {
                if (true || DEBUG) Slog.v(TAG, "Adding window token: " + mCurToken);
                mIWindowManager.addWindowToken(mCurToken,
                        WindowManager.LayoutParams.TYPE_INPUT_METHOD);
            } catch (RemoteException e) {
            }
            return new InputBindResult(null, null, mCurId, mCurSeq,
                    mCurUserActionNotificationSequenceNumber);
        } else {
            mCurIntent = null;
            Slog.w(TAG, "Failure connecting to input method service: "
                    + mCurIntent);
        }
        return null;
    }


    private boolean bindCurrentInputMethodService(
            Intent service, ServiceConnection conn, int flags) {
        if (service == null || conn == null) {
            Slog.e(TAG, "--- bind failed: service = " + service + ", conn = " + conn);
            return false;
        }
        return mContext.bindServiceAsUser(service, conn, flags,
                new UserHandle(mSettings.getCurrentUserId()));
    }

// AbstractInputMethodService.java
    @Override
    final public IBinder onBind(Intent intent) {
        if (mInputMethod == null) {
            mInputMethod = onCreateInputMethodInterface();
        }
        // IInputMethodWrapper 将 IMMS的调用转化为 message ，
        // 然后在 message 线程调用 mInputMethod 对应的接口，
        // 实现输入法的异步处理
```

```
            return new IInputMethodWrapper(this, mInputMethod);
        }

// InputMethodService.java
    /**
     * Implement to return our standard {@link InputMethodImpl}.  Subclasses
     * can override to provide their own customized version.
     */
    @Override
    public AbstractInputMethodImpl onCreateInputMethodInterface() {
        return new InputMethodImpl();
    }

// 由于IMMS是以bindService的方式启动输入法service，所以当输入法service启动完
// 成后它就会回调IMMS的onServiceConnected
// InputMethodManagerService.java
    @Override
    public void onServiceConnected(ComponentName name, IBinder service) {
        synchronized (mMethodMap) {
            if (mCurIntent != null && name.equals(mCurIntent.getComponent())) {
                // 保存输入法Service 传递过来的 通信接口IInputMethod
                mCurMethod = IInputMethod.Stub.asInterface(service);
                if (mCurToken == null) {
                    Slog.w(TAG, "Service connected without a token!");
                    unbindCurrentMethodLocked(false);
                    return;
                }
                if (DEBUG) Slog.v(TAG, "Initiating attach with token: " + mCurToken);
                // 将刚刚创建的window token传递给输入法service,然后输入用这个token
                // 创建window,这样IMMS可以用根据这个token找到输入法在IMMS里
                // 的数据及输入法window在WMS里的数据
                executeOrSendMessage(mCurMethod, mCaller.obtainMessageOO(
                        MSG_ATTACH_TOKEN, mCurMethod, mCurToken));
                if (mCurClient != null) {
                    clearClientSessionLocked(mCurClient);
                    // 请求为程序和输入法建立一个连接会话，这样client就可以直接和
                    // 输入法通信了
                    requestClientSessionLocked(mCurClient);
                }
            }
        }
    }


    case MSG_ATTACH_TOKEN:
        args = (SomeArgs)msg.obj;
        try {
            if (DEBUG) Slog.v(TAG, "Sending attach of token: " + args.arg2);
            // 和输入法通信
            ((IInputMethod)args.arg1).attachToken((IBinder)args.arg2);
        } catch (RemoteException e) {
        }
        args.recycle();
        return true;

// InputMethodService.java

    /**
     * Concrete implementation of
     * {@link AbstractInputMethodService.AbstractInputMethodImpl} that provides
     * all of the standard behavior for an input method.
     */
    public class InputMethodImpl extends AbstractInputMethodImpl {
        /**
         * Take care of attaching the given window token provided by the system.
         */
```

```
        public void attachToken(IBinder token) {
            if (mToken == null) {
                // 保存token
                mToken = token;
                // 这样输入法的window就绑定这个window token
                mWindow.setToken(token);
            }
        }
    }

// InputMethodManagerService.java
    void requestClientSessionLocked(ClientState cs) {
        if (!cs.sessionRequested) {
            if (DEBUG) Slog.v(TAG, "Creating new session for client " + cs);
            // 这里又出现了InputChannel对，很面熟吧，在前面几篇文章已经详细分析过
            // 了，可见它已经成为一种通用的跨平台的数据通信接口了
            InputChannel[] channels = InputChannel.openInputChannelPair(cs.toString());
            cs.sessionRequested = true;
            executeOrSendMessage(mCurMethod, mCaller.obtainMessageOOO(
                    MSG_CREATE_SESSION, mCurMethod, channels[1],
                    new MethodCallback(this, mCurMethod, channels[0])));
        }
    }

    case MSG_CREATE_SESSION: {
        args = (SomeArgs)msg.obj;
        IInputMethod method = (IInputMethod)args.arg1;
        InputChannel channel = (InputChannel)args.arg2;
        try {
            method.createSession(channel, (IInputSessionCallback)args.arg3);
        } catch (RemoteException e) {
        } finally {
            // Dispose the channel if the input method is not local to this process
            // because the remote proxy will get its own copy when unparceled.
            if (channel != null && Binder.isProxy(method)) {
                channel.dispose();
            }
        }
        args.recycle();
        return true;
    }

//上面是IMMS端，下面就看IMS输入法端的处理
// IInputMethodWrapper.java
    @Override
    public void createSession(InputChannel channel, IInputSessionCallback callback) {
        mCaller.executeOrSendMessage(mCaller.obtainMessageOO(DO_CREATE_SESSION,
                channel, callback));
    }

    case DO_CREATE_SESSION: {
        SomeArgs args = (SomeArgs)msg.obj;
        inputMethod.createSession(new InputMethodSessionCallbackWrapper(
                mContext, (InputChannel)args.arg1,
                (IInputSessionCallback)args.arg2));
        args.recycle();
        return;
    }

// AbstractInputMethodService.java
    /**
   * Base class for derived classes to implement their {@link InputMethod}
   * interface.  This takes care of basic maintenance of the input method,
   * but most behavior must be implemented in a derived class.
   */
    public abstract class AbstractInputMethodImpl implements InputMethod {
        /**
```

```
     * Instantiate a new client session for the input method, by calling
     * back to {@link AbstractInputMethodService#onCreateInputMethodSessionInterface()
     * AbstractInputMethodService.onCreateInputMethodSessionInterface()}.
     */
        public void createSession(SessionCallback callback) {
            callback.sessionCreated(onCreateInputMethodSessionInterface());
        }
    }

// InputMethodManagerService.java
        @Override
        public void sessionCreated(IInputMethodSession session) {
            long ident = Binder.clearCallingIdentity();
            try {
                mParentIMMS.onSessionCreated(mMethod, session, mChannel);
            } finally {
                Binder.restoreCallingIdentity(ident);
            }
        }
    }

    void onSessionCreated(IInputMethod method, IInputMethodSession session,
            InputChannel channel) {
        synchronized (mMethodMap) {
            if (mCurMethod != null && method != null
                    && mCurMethod.asBinder() == method.asBinder()) {
                if (mCurClient != null) {
                    clearClientSessionLocked(mCurClient);
                    mCurClient.curSession = new SessionState(mCurClient,
                            method, session, channel);
                    InputBindResult res = attachNewInputLocked(true);
                    if (res.method != null) {
                        executeOrSendMessage(mCurClient.client, mCaller.obtainMessageOO(
                                MSG_BIND_CLIENT, mCurClient.client, res));
                    }
                    return;
                }
            }
        }

        // Session abandoned.  Close its associated input channel.
        channel.dispose();
    }

    //输入法和view绑定
    InputBindResult attachNewInputLocked(boolean initial) {
        if (!mBoundToMethod) {
            executeOrSendMessage(mCurMethod, mCaller.obtainMessageOO(
                    MSG_BIND_INPUT, mCurMethod, mCurClient.binding));
            mBoundToMethod = true;
        }
        final SessionState session = mCurClient.curSession;
        if (initial) {
            executeOrSendMessage(session.method, mCaller.obtainMessageIOOO(
                    MSG_START_INPUT, mCurInputContextMissingMethods, session, mCurInputContext,
                    mCurAttribute));
        } else {
            executeOrSendMessage(session.method, mCaller.obtainMessageIOOO(
                    MSG_RESTART_INPUT, mCurInputContextMissingMethods, session, mCurInputContext,
                    mCurAttribute));
        }
        if (mShowRequested) {
            if (DEBUG) Slog.v(TAG, "Attach new input asks to show input");
            showCurrentInputLocked(getAppShowFlags(), null);
        }
        return new InputBindResult(session.session,
                (session.channel != null ? session.channel.dup() : null),
```

```
                mCurId, mCurSeq, mCurUserActionNotificationSequenceNumber);
    }


    case MSG_BIND_INPUT:
        args = (SomeArgs)msg.obj;
        try {
            ((IInputMethod)args.arg1).bindInput((InputBinding)args.arg2);
        } catch (RemoteException e) {
        }
        args.recycle();
        return true;

    case MSG_START_INPUT: {
        int missingMethods = msg.arg1;
        args = (SomeArgs) msg.obj;
        try {
            SessionState session = (SessionState) args.arg1;
            setEnabledSessionInMainThread(session);
            session.method.startInput((IInputContext) args.arg2, missingMethods,
                    (EditorInfo) args.arg3);
        } catch (RemoteException e) {
        }
        args.recycle();
        return true;
    }

    case MSG_BIND_CLIENT: {
        args = (SomeArgs)msg.obj;
        IInputMethodClient client = (IInputMethodClient)args.arg1;
        InputBindResult res = (InputBindResult)args.arg2;
        try {
            // 调回到程序端InputMethodManager.onBindMethod()
            client.onBindMethod(res);
        } catch (RemoteException e) {
            Slog.w(TAG, "Client died receiving input method " + args.arg2);
        } finally {
            // Dispose the channel if the input method is not local to this process
            // because the remote proxy will get its own copy when unparceled.
            if (res.channel != null && Binder.isProxy(client)) {
                res.channel.dispose();
            }
        }
        args.recycle();
        return true;
    }


// IInputMethodWrapper.java
    @Override
    public void startInput(IInputContext inputContext,
            @InputConnectionInspector.MissingMethodFlags final int missingMethods,
            EditorInfo attribute) {
        mCaller.executeOrSendMessage(mCaller.obtainMessageIOO(DO_START_INPUT,
                missingMethods, inputContext, attribute));
    }

    case DO_START_INPUT: {
        SomeArgs args = (SomeArgs)msg.obj;
        int missingMethods = msg.arg1;
        // IInputContext就是输入法和文本输入view的通信接口
        // 通过这个接口，输入法能够获取view的信息，也能够直接将文本传送给view
        IInputContext inputContext = (IInputContext)args.arg1;
        InputConnection ic = inputContext != null
                ? new InputConnectionWrapper(inputContext, missingMethods) : null;
        EditorInfo info = (EditorInfo)args.arg2;
        info.makeCompatible(mTargetSdkVersion);
        inputMethod.startInput(ic, info);
                    1 ()
```

```
            args.recycle();
            return;
        }

    // InputMethodService.java
        public void startInput(InputConnection ic, EditorInfo attribute) {
            if (DEBUG) Log.v(TAG, "startInput(): editor=" + attribute);
            doStartInput(ic, attribute, false);
        }

        void doStartInput(InputConnection ic, EditorInfo attribute, boolean restarting) {
            if (!restarting) {
                doFinishInput();
            }
            mInputStarted = true;
            mStartedInputConnection = ic;
            mInputEditorInfo = attribute;
            initialize();
            if (DEBUG) Log.v(TAG, "CALL: onStartInput");
            onStartInput(attribute, restarting);
            if (mWindowVisible) {
                if (mShowInputRequested) {
                    if (DEBUG) Log.v(TAG, "CALL: onStartInputView");
                    mInputViewStarted = true;
                    onStartInputView(mInputEditorInfo, restarting);
                    startExtractingText(true);
                } else if (mCandidatesVisibility == View.VISIBLE) {
                    if (DEBUG) Log.v(TAG, "CALL: onStartCandidatesView");
                    mCandidatesViewStarted = true;
                    onStartCandidatesView(mInputEditorInfo, restarting);
                }
            }
        }
```

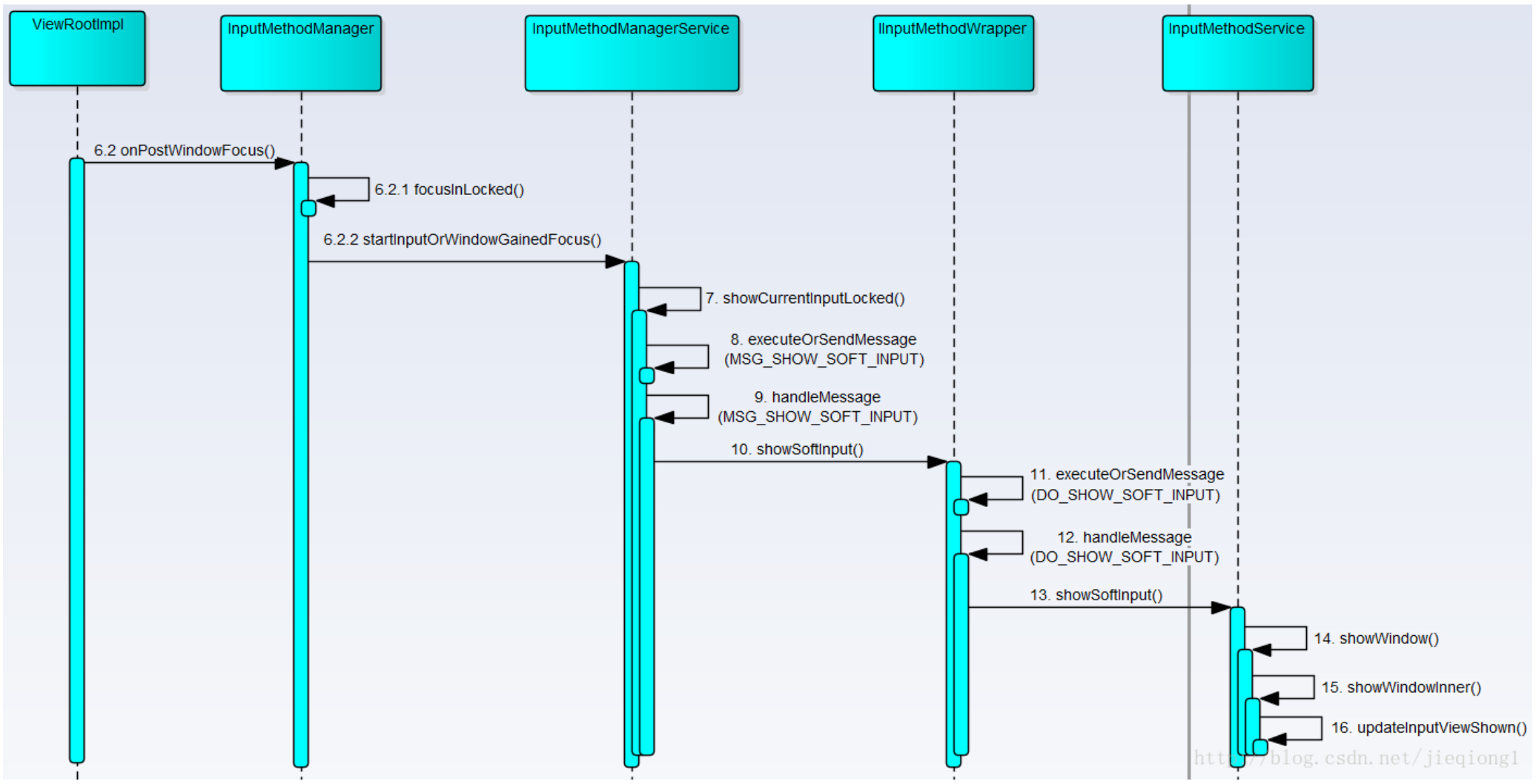
到此焦点view已经通过调用IMMS的startInput和输入法绑定了，但是此时输入法还没有显示。但是系统紧接着会调用windowGainFocus来显示输入法。

6.2 onPostWindowFocus() 之后的逻辑下次再写

二. 显示输入法

6.2 onPostWindowFocus() 之后的流程

下边的时序图不太清晰，csdn上传了下 原图：http://download.csdn.net/detail/jieqiong1/9836368




```
// InputMethodManager.java
/**
* Called by ViewAncestor when its window gets input focus.
* @hide
*/
public void onPostWindowFocus(View rootView, View focusedView, int softInputMode,
        boolean first, int windowFlags) {
    boolean forceNewFocus = false;
    synchronized (mH) {
        if (DEBUG) Log.v(TAG, "onWindowFocus: " + focusedView
                + " softInputMode=" + softInputMode
                + " first=" + first + " flags=#"
                + Integer.toHexString(windowFlags));
        if (mHasBeenInactive) {
            if (DEBUG) Log.v(TAG, "Has been inactive!  Starting fresh");
            mHasBeenInactive = false;
            forceNewFocus = true;
        }
        // view 获取焦点
        focusInLocked(focusedView != null ? focusedView : rootView);
    }

    int controlFlags = 0;
    if (focusedView != null) {
        controlFlags |= CONTROL_WINDOW_VIEW_HAS_FOCUS;
        if (focusedView.onCheckIsTextEditor()) {
            controlFlags |= CONTROL_WINDOW_IS_TEXT_EDITOR;
        }
    }
    if (first) {
        controlFlags |= CONTROL_WINDOW_FIRST;
    }

    // 确认当前 focused view 是否已经调用过 startInputInner() 来绑定输入法，
    // 因为前面 mView.dispatchWindowFocusChanged() 已经完成了 focused view 的绑定，
    // 大部分情况下，该函数返回 false，不会再次调用  startInputInner()
    if (checkFocusNoStartInput(forceNewFocus)) {
        // We need to restart input on the current focus view.  This
        // should be done in conjunction with telling the system service

        // ...
```

```
            // about the window gaining focus, to help make the transition
            // smooth.
            if (startInputInner(InputMethodClient.START_INPUT_REASON_WINDOW_FOCUS_GAIN,
                    rootView.getWindowToken(), controlFlags, softInputMode, windowFlags)) {
                return;
            }
        }

        // For some reason we didn't do a startInput + windowFocusGain, so
        // we'll just do a window focus gain and call it a day.
        synchronized (mH) {
            try {
                if (DEBUG) Log.v(TAG, "Reporting focus gain, without startInput");
                // //调用IMMS windowGainedFocus函数
                mService.startInputOrWindowGainedFocus(
                        InputMethodClient.START_INPUT_REASON_WINDOW_FOCUS_GAIN_REPORT_ONLY, mClient,
                        rootView.getWindowToken(), controlFlags, softInputMode, windowFlags, null,
                        null, 0 /* missingMethodFlags */);
            } catch (RemoteException e) {
                throw e.rethrowFromSystemServer();
            }
        }
    }

    void focusInLocked(View view) {
        if (DEBUG) Log.v(TAG, "focusIn: " + dumpViewInfo(view));

        if (view != null && view.isTemporarilyDetached()) {
            // This is a request from a view that is temporarily detached from a window.
            if (DEBUG) Log.v(TAG, "Temporarily detached view, ignoring");
            return;
        }

        if (mCurRootView != view.getRootView()) {
            // This is a request from a window that isn't in the window with
            // IME focus, so ignore it.
            if (DEBUG) Log.v(TAG, "Not IME target window, ignoring");
            return;
        }

        mNextServedView = view;
        scheduleCheckFocusLocked(view);
    }


// InputMethodManagerService.java
    @Override
    public InputBindResult startInputOrWindowGainedFocus(
            /* @InputMethodClient.StartInputReason */ final int startInputReason,
            IInputMethodClient client, IBinder windowToken, int controlFlags, int softInputMode,
            int windowFlags, @Nullable EditorInfo attribute, IInputContext inputContext,
            /* @InputConnectionInspector.missingMethods */ final int missingMethods) {
        if (windowToken != null) {
            return windowGainedFocus(startInputReason, client, windowToken, controlFlags,
                    softInputMode, windowFlags, attribute, inputContext, missingMethods);
        } else {
            return startInput(startInputReason, client, inputContext, missingMethods, attribute,
                    controlFlags);
        }
    }


    private InputBindResult windowGainedFocus(
            /* @InputMethodClient.StartInputReason */ final int startInputReason,
            IInputMethodClient client, IBinder windowToken, int controlFlags, int softInputMode,
            int windowFlags, EditorInfo attribute, IInputContext inputContext,
```

```
        /* @InputConnectionInspector.MissingMethods */ final int missingMethods) {
    // Needs to check the validity before clearing calling identity
    final boolean calledFromValidUser = calledFromValidUser();
    InputBindResult res = null;
    long ident = Binder.clearCallingIdentity();
    try {
        synchronized (mMethodMap) {
            ...
            mCurFocusedWindow = windowToken;
            mCurFocusedWindowClient = cs;

            // Should we auto-show the IME even if the caller has not
            // specified what should be done with it?
            // We only do this automatically if the window can resize
            // to accommodate the IME (so what the user sees will give
            // them good context without input information being obscured
            // by the IME) or if running on a large screen where there
            // is more room for the target window + IME.
            final boolean doAutoShow =
                    (softInputMode & WindowManager.LayoutParams.SOFT_INPUT_MASK_ADJUST)
                            == WindowManager.LayoutParams.SOFT_INPUT_ADJUST_RESIZE
                    || mRes.getConfiguration().isLayoutSizeAtLeast(
                            Configuration.SCREENLAYOUT_SIZE_LARGE);
            final boolean isTextEditor =
                    (controlFlags&InputMethodManager.CONTROL_WINDOW_IS_TEXT_EDITOR) != 0;

            // We want to start input before showing the IME, but after closing
            // it.  We want to do this after closing it to help the IME disappear
            // more quickly (not get stuck behind it initializing itself for the
            // new focused input, even if its window wants to hide the IME).
            boolean didStart = false;

            switch (softInputMode&WindowManager.LayoutParams.SOFT_INPUT_MASK_STATE) {
                case WindowManager.LayoutParams.SOFT_INPUT_STATE_UNSPECIFIED:
                    if (!isTextEditor || !doAutoShow) {
                        if (WindowManager.LayoutParams.mayUseInputMethod(windowFlags)) {
                            // There is no focus view, and this window will
                            // be behind any soft input window, so hide the
                            // soft input window if it is shown.
                            if (DEBUG) Slog.v(TAG, "Unspecified window will hide input");
                            hideCurrentInputLocked(InputMethodManager.HIDE_NOT_ALWAYS, null);
                        }
                    } else if (isTextEditor && doAutoShow && (softInputMode &
                            WindowManager.LayoutParams.SOFT_INPUT_IS_FORWARD_NAVIGATION) != 0) {
                        // There is a focus view, and we are navigating forward
                        // into the window, so show the input window for the user.
                        // We only do this automatically if the window can resize
                        // to accommodate the IME (so what the user sees will give
                        // them good context without input information being obscured
                        // by the IME) or if running on a large screen where there
                        // is more room for the target window + IME.
                        if (DEBUG) Slog.v(TAG, "Unspecified window will show input");
                        if (attribute != null) {
                            res = startInputUncheckedLocked(cs, inputContext,
                                    missingMethods, attribute, controlFlags);
                            didStart = true;
                        }
                        // 调用 showCurrentInputLocked()
                        showCurrentInputLocked(InputMethodManager.SHOW_IMPLICIT, null);
                    }
                    break;
                case WindowManager.LayoutParams.SOFT_INPUT_STATE_UNCHANGED:
                    // Do nothing.
                    break;
                case WindowManager.LayoutParams.SOFT_INPUT_STATE_HIDDEN:
                    if ((softInputMode &
```

```
                        WindowManager.LayoutParams.SOFT_INPUT_IS_FORWARD_NAVIGATION) != 0) {
                    if (DEBUG) Slog.v(TAG, "Window asks to hide input going forward");
                    hideCurrentInputLocked(0, null);
                }
                break;
            case WindowManager.LayoutParams.SOFT_INPUT_STATE_ALWAYS_HIDDEN:
                if (DEBUG) Slog.v(TAG, "Window asks to hide input");
                hideCurrentInputLocked(0, null);
                break;
            case WindowManager.LayoutParams.SOFT_INPUT_STATE_VISIBLE:
                if ((softInputMode &
                        WindowManager.LayoutParams.SOFT_INPUT_IS_FORWARD_NAVIGATION) != 0) {
                    if (DEBUG) Slog.v(TAG, "Window asks to show input going forward");
                    if (attribute != null) {
                        res = startInputUncheckedLocked(cs, inputContext,
                                missingMethods, attribute, controlFlags);
                        didStart = true;
                    }
                    showCurrentInputLocked(InputMethodManager.SHOW_IMPLICIT, null);
                }
                break;
            case WindowManager.LayoutParams.SOFT_INPUT_STATE_ALWAYS_VISIBLE:
                if (DEBUG) Slog.v(TAG, "Window asks to always show input");
                if (attribute != null) {
                    res = startInputUncheckedLocked(cs, inputContext, missingMethods,
                            attribute, controlFlags);
                    didStart = true;
                }
                showCurrentInputLocked(InputMethodManager.SHOW_IMPLICIT, null);
                break;
        }

        if (!didStart && attribute != null) {
            res = startInputUncheckedLocked(cs, inputContext, missingMethods, attribute,
                    controlFlags);
        }
    }
} finally {
    Binder.restoreCallingIdentity(ident);
}

return res;
}

boolean showCurrentInputLocked(int flags, ResultReceiver resultReceiver) {
mShowRequested = true;
if (mAccessibilityRequestingNoSoftKeyboard) {
    return false;
}

if ((flags&InputMethodManager.SHOW_FORCED) != 0) {
    mShowExplicitlyRequested = true;
    mShowForced = true;
} else if ((flags&InputMethodManager.SHOW_IMPLICIT) == 0) {
    mShowExplicitlyRequested = true;
}

if (!mSystemReady) {
    return false;
}

boolean res = false;
if (mCurMethod != null) {
    if (DEBUG) Slog.d(TAG, "showCurrentInputLocked: mCurToken=" + mCurToken);
    // 发消息 MSG_SHOW_SOFT_INPUT
    executeOrSendMessage(mCurMethod, mCaller.obtainMessageIOO(
```

```
                MSG_SHOW_SOFT_INPUT, getImeShowFlags(), mCurMethod,
                resultReceiver));
        mInputShown = true;
        if (mHaveConnection && !mVisibleBound) {
            bindCurrentInputMethodService(
                    mCurIntent, mVisibleConnection, Context.BIND_AUTO_CREATE
                            | Context.BIND_TREAT_LIKE_ACTIVITY
                            | Context.BIND_FOREGROUND_SERVICE);
            mVisibleBound = true;
        }
        res = true;
    } else if (mHaveConnection && SystemClock.uptimeMillis()
            >= (mLastBindTime+TIME_TO_RECONNECT)) {
        // The client has asked to have the input method shown, but
        // we have been sitting here too long with a connection to the
        // service and no interface received, so let's disconnect/connect
        // to try to prod things along.
        EventLog.writeEvent(EventLogTags.IMF_FORCE_RECONNECT_IME, mCurMethodId,
                SystemClock.uptimeMillis()-mLastBindTime,1);
        Slog.w(TAG, "Force disconnect/connect to the IME in showCurrentInputLocked()");
        mContext.unbindService(this);
        bindCurrentInputMethodService(mCurIntent, this, Context.BIND_AUTO_CREATE
                | Context.BIND_NOT_VISIBLE);
    } else {
        if (DEBUG) {
            Slog.d(TAG, "Can't show input: connection = " + mHaveConnection + ", time = "
                    + ((mLastBindTime+TIME_TO_RECONNECT) - SystemClock.uptimeMillis()));
        }
    }

    return res;
}


case MSG_SHOW_SOFT_INPUT:
    args = (SomeArgs)msg.obj;
    try {
        if (DEBUG) Slog.v(TAG, "Calling " + args.arg1 + ".showSoftInput("
                + msg.arg1 + ", " + args.arg2 + ")");
        // IInputMethod.showSoftInput() 即 IInputMethodWrapper.showSoftInput()
        ((IInputMethod)args.arg1).showSoftInput(msg.arg1, (ResultReceiver)args.arg2);
    } catch (RemoteException e) {
    }
    args.recycle();
    return true;


// IInputMethodWrapper.java
    @Override
    public void showSoftInput(int flags, ResultReceiver resultReceiver) {
        mCaller.executeOrSendMessage(mCaller.obtainMessageIO(DO_SHOW_SOFT_INPUT,
                flags, resultReceiver));
    }

    case DO_SHOW_SOFT_INPUT:
        // 这个inputMethod是通过onCreateInputMethodInterface函数创建的
        // InputMethodImpl对象
        inputMethod.showSoftInput(msg.arg1, (ResultReceiver)msg.obj);
        return;


// InputMethodService.InputMethodImpl.showSoftInput()
        /**
     * Handle a request by the system to show the soft input area.
     */
        public void showSoftInput(int flags, ResultReceiver resultReceiver) {
            if (DEBUG) Log.v(TAG, "showSoftInput()");
```

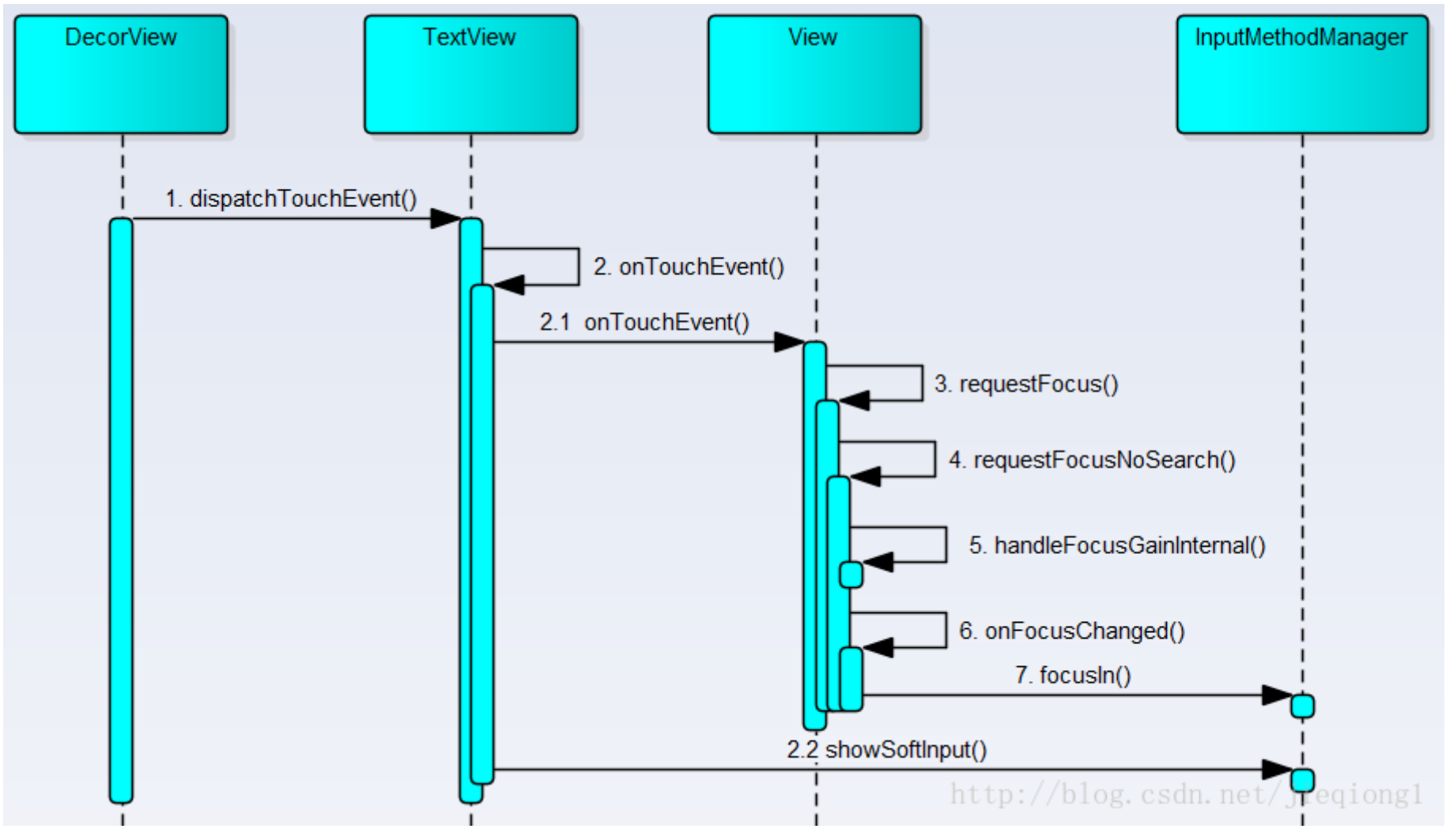
```
            boolean wasVis = isInputViewShown();
            if (dispatchOnShowInputRequested(flags, false)) {
                try {
                    // 这个是真正显示UI的函数
                    showWindow(true);
                } catch (BadTokenException e) {
                    // We have ignored BadTokenException here since Jelly Bean MR-2 (API Level 18).
                    // We could ignore BadTokenException in InputMethodService#showWindow() instead,
                    // but it may break assumptions for those who override #showWindow() that we can
                    // detect errors in #showWindow() by checking BadTokenException.
                    // TODO: Investigate its feasibility.  Update JavaDoc of #showWindow() of
                    // whether it's OK to override #showWindow() or not.
                }
            }
            clearInsetOfPreviousIme();
            // If user uses hard keyboard, IME button should always be shown.
            boolean showing = isInputViewShown();
            mImm.setImeWindowStatus(mToken, IME_ACTIVE | (showing ? IME_VISIBLE : 0),
                    mBackDisposition);
            if (resultReceiver != null) {
                resultReceiver.send(wasVis != isInputViewShown()
                        ? InputMethodManager.RESULT_SHOWN
                        : (wasVis ? InputMethodManager.RESULT_UNCHANGED_SHOWN
                                : InputMethodManager.RESULT_UNCHANGED_HIDDEN), null);
            }
        }



    public void showWindow(boolean showInput) {
        if (DEBUG) Log.v(TAG, "Showing window: showInput=" + showInput
                + " mShowInputRequested=" + mShowInputRequested
                + " mWindowAdded=" + mWindowAdded
                + " mWindowCreated=" + mWindowCreated
                + " mWindowVisible=" + mWindowVisible
                + " mInputStarted=" + mInputStarted
                + " mShowInputFlags=" + mShowInputFlags);

        if (mInShowWindow) {
            Log.w(TAG, "Re-entrance in to showWindow");
            return;
        }

        try {
            mWindowWasVisible = mWindowVisible;
            mInShowWindow = true;
            // 调用 showWindowInner()
            showWindowInner(showInput);
        } catch (BadTokenException e) {
            // BadTokenException is a normal consequence in certain situations, e.g., swapping IMEs
            // while there is a DO_SHOW_SOFT_INPUT message in the IIMethodWrapper queue.
            if (DEBUG) Log.v(TAG, "BadTokenException: IME is done.");
            mWindowVisible = false;
            mWindowAdded = false;
            // Rethrow the exception to preserve the existing behavior.  Some IMEs may have directly
            // called this method and relied on this exception for some clean-up tasks.
            // TODO: Give developers a clear guideline of whether it's OK to call this method or
            // InputMethodManager#showSoftInputFromInputMethod() should always be used instead.
            throw e;
        } finally {
            // TODO: Is it OK to set true when we get BadTokenException?
            mWindowWasVisible = true;
            mInShowWindow = false;
        }
    }
```

```
382         void showWindowInner(boolean showInput) {
383             boolean doShowInput = false;
384             final int previousImeWindowStatus =
385                     (mWindowVisible ? IME_ACTIVE : 0) | (isInputViewShown() ? IME_VISIBLE : 0);
386             mWindowVisible = true;
387             if (!mShowInputRequested && mInputStarted && showInput) {
388                 doShowInput = true;
389                 mShowInputRequested = true;
390             }
391 
392             if (DEBUG) Log.v(TAG, "showWindow: updating UI");
393             initialize();
394             updateFullscreenMode();
395             //这个函数会创建输入法的键盘
396             updateInputViewShown();
397 
398             if (!mWindowAdded || !mWindowCreated) {
399                 mWindowAdded = true;
400                 mWindowCreated = true;
401                 initialize();
402                 if (DEBUG) Log.v(TAG, "CALL: onCreateCandidatesView");
403                 //创建输入法dialog里的词条选择View
404                 View v = onCreateCandidatesView();
405                 if (DEBUG) Log.v(TAG, "showWindow: candidates=" + v);
406                 if (v != null) {
407                     setCandidatesView(v);
408                 }
409             }
410             if (mShowInputRequested) {
411                 if (!mInputViewStarted) {
412                     if (DEBUG) Log.v(TAG, "CALL: onStartInputView");
413                     mInputViewStarted = true;
414                     onStartInputView(mInputEditorInfo, false);
415                 }
416             } else if (!mCandidatesViewStarted) {
417                 if (DEBUG) Log.v(TAG, "CALL: onStartCandidatesView");
418                 mCandidatesViewStarted = true;
419                 onStartCandidatesView(mInputEditorInfo, false);
420             }
421 
422             if (doShowInput) {
423                 startExtractingText(false);
424             }
425 
426             final int nextImeWindowStatus = IME_ACTIVE | (isInputViewShown() ? IME_VISIBLE : 0);
427             if (previousImeWindowStatus != nextImeWindowStatus) {
428                 mImm.setImeWindowStatus(mToken, nextImeWindowStatus, mBackDisposition);
429             }
430             if ((previousImeWindowStatus & IME_ACTIVE) == 0) {
431                 if (DEBUG) Log.v(TAG, "showWindow: showing!");
432                 onWindowShown();
433                 //这个是输入法Dialog的window,这里开始就显示UI了
434                 mWindow.show();
435                 // Put here rather than in onWindowShown() in case people forget to call
436                 // super.onWindowShown().
437                 mShouldClearInsetOfPreviousIme = false;
438             }
439         }
440 
441 
442         /**
443          * Re-evaluate whether the soft input area should currently be shown, and
444          * update its UI if this has changed since the last time it
445          * was evaluated.  This will call {@link #onEvaluateInputViewShown()} to
446          * determine whether the input view should currently be shown.  You
447          * can use {@link #isInputViewShown()} to determine if the input view
448          * is currently shown.
449          */
```

```
     */
    public void updateInputViewShown() {
        boolean isShown = mShowInputRequested && onEvaluateInputViewShown();
        if (mIsInputViewShown != isShown && mWindowVisible) {
            mIsInputViewShown = isShown;
            mInputFrame.setVisibility(isShown ? View.VISIBLE : View.GONE);
            if (mInputView == null) {
                initialize();
                // 这个是核心view，创建显示键盘的根view
                View v = onCreateInputView();
                if (v != null) {
                    setInputView(v);
                }
            }
        }
    }
```

5. 用户单击输入框显示输入法

http://blog.csdn.net/huangyabin001/article/details/28435093 中 作者从 InputEventReceiver.dispatchInputEvent()开始分析的，本文从TextView.onTouchEvent()开始写。




```
// TextView.java
    @Override
    public boolean onTouchEvent(MotionEvent event) {
        final int action = event.getActionMasked();
        if (mEditor != null) {
            mEditor.onTouchEvent(event);

            if (mEditor.mSelectionModifierCursorController != null &&
                    mEditor.mSelectionModifierCursorController.isDragAcceleratorActive()) {
                return true;
            }
        }
        //非 EditText 只会执行 View.onTouchEvent(),该函数是另一种将view和输入法绑定的调用
        //而 EditText 会调用 imm.showSoftInput() 会显示输入法
        final boolean superResult = super.onTouchEvent(event);

        /*
     * Don't handle the release after a long press, because it will move the selection away from
```

```
 * whatever the menu action was trying to affect. If the long press should have triggered an
 * insertion action mode, we can now actually show it.
 */
    if (mEditor != null && mEditor.mDiscardNextActionUp && action == MotionEvent.ACTION_UP) {
        mEditor.mDiscardNextActionUp = false;

        if (mEditor.mIsInsertionActionModeStartPending) {
            mEditor.startInsertionActionMode();
            mEditor.mIsInsertionActionModeStartPending = false;
        }
        return superResult;
    }

    final boolean touchIsFinished = (action == MotionEvent.ACTION_UP) &&
            (mEditor == null || !mEditor.mIgnoreActionUpEvent) && isFocused();

     if ((mMovement != null || onCheckIsTextEditor()) && isEnabled()
            && mText instanceof Spannable && mLayout != null) {
        boolean handled = false;

        if (mMovement != null) {
            handled |= mMovement.onTouchEvent(this, (Spannable) mText, event);
        }

        final boolean textIsSelectable = isTextSelectable();
        if (touchIsFinished && mLinksClickable && mAutoLinkMask != 0 && textIsSelectable) {
            // The LinkMovementMethod which should handle taps on links has not been installed
            // on non editable text that support text selection.
            // We reproduce its behavior here to open links for these.
            ClickableSpan[] links = ((Spannable) mText).getSpans(getSelectionStart(),
                    getSelectionEnd(), ClickableSpan.class);

            if (links.length > 0) {
                links[0].onClick(this);
                handled = true;
            }
        }

        if (touchIsFinished && (isTextEditable() || textIsSelectable)) {
            // Show the IME, except when selecting in read-only text.
            final InputMethodManager imm = InputMethodManager.peekInstance();
            viewClicked(imm);
            // 这个是真正显示输入法的调用
            if (!textIsSelectable && mEditor.mShowSoftInputOnFocus) {
                handled |= imm != null && imm.showSoftInput(this, 0);
                }

                // The above condition ensures that the mEditor is not null
                mEditor.onTouchUpEvent(event);

                handled = true;
            }

            if (handled) {
                return true;
            }
        }

        return superResult;
    }

// View.java
    /**
   * Implement this method to handle touch screen motion events.
   * <p>
   * If this method is used to detect click actions, it is recommended that
   * the actions be performed by implementing and calling
```

```
* {@link #performClick()}. This will ensure consistent system behavior,
* including:
* <ul>
* <li>obeying click sound preferences
* <li>dispatching OnClickListener calls
* <li>handling {@link AccessibilityNodeInfo#ACTION_CLICK ACTION_CLICK} when
* accessibility features are enabled
* </ul>
*
* @param event The motion event.
* @return True if the event was handled, false otherwise.
*/
  public boolean onTouchEvent(MotionEvent event) {
      final float x = event.getX();
      final float y = event.getY();
      final int viewFlags = mViewFlags;
      final int action = event.getAction();

      ...

      if (((viewFlags & CLICKABLE) == CLICKABLE ||
              (viewFlags & LONG_CLICKABLE) == LONG_CLICKABLE) ||
          (viewFlags & CONTEXT_CLICKABLE) == CONTEXT_CLICKABLE) {
      switch (action) {
          case MotionEvent.ACTION_UP:
              boolean prepressed = (mPrivateFlags & PFLAG_PREPRESSED) != 0;
              if ((mPrivateFlags & PFLAG_PRESSED) != 0 || prepressed) {
                  // take focus if we don't have it already and we should in
                  // touch mode.
                  boolean focusTaken = false;
                  //让view获得焦点
                  if (isFocusable() && isFocusableInTouchMode() && !isFocused()) {
                      focusTaken = requestFocus();
                  }

                  ...
              }
              mIgnoreNextUpEvent = false;
              break;

          ...
      }

      return true;
  }
      return false;
  }

  /**
* Call this to try to give focus to a specific view or to one of its descendants
* and give it hints about the direction and a specific rectangle that the focus
* is coming from.  The rectangle can help give larger views a finer grained hint
* about where focus is coming from, and therefore, where to show selection, or
* forward focus change internally.
*
* A view will not actually take focus if it is not focusable ({@link #isFocusable} returns
* false), or if it is focusable and it is not focusable in touch mode
* ({@link #isFocusableInTouchMode}) while the device is in touch mode.
*
* A View will not take focus if it is not visible.
*
* A View will not take focus if one of its parents has
* {@link android.view.ViewGroup#getDescendantFocusability()} equal to
* {@link ViewGroup#FOCUS_BLOCK_DESCENDANTS}.
*
* See also {@link #focusSearch(int)}, which is what you call to say that you
```

```
     * have focus, and you want your parent to look for the next one.
     *
     * You may wish to override this method if your custom {@link View} has an internal
     * {@link View} that it wishes to forward the request to.
     *
     * @param direction One of FOCUS_UP, FOCUS_DOWN, FOCUS_LEFT, and FOCUS_RIGHT
     * @param previouslyFocusedRect The rectangle (in this View's coordinate system)
     *       to give a finer grained hint about where focus is coming from.  May be null
     *       if there is no hint.
     * @return Whether this view or one of its descendants actually took focus.
     */
    public boolean requestFocus(int direction, Rect previouslyFocusedRect) {
        return requestFocusNoSearch(direction, previouslyFocusedRect);
    }

    private boolean requestFocusNoSearch(int direction, Rect previouslyFocusedRect) {
        // need to be focusable
        // 该view必须是可以获取焦点的
        if ((mViewFlags & FOCUSABLE_MASK) != FOCUSABLE ||
                (mViewFlags & VISIBILITY_MASK) != VISIBLE) {
            return false;
        }

        // need to be focusable in touch mode if in touch mode
        if (isInTouchMode() &&
            (FOCUSABLE_IN_TOUCH_MODE != (mViewFlags & FOCUSABLE_IN_TOUCH_MODE))) {
               return false;
        }

        // 检查的是属性：android:descendantFocusability=”blocksDescendants”，
        // 这个属性可以解决 listView 等容器类View没法获取点击事件问题,
        // 当父亲设置了这个属性, 子view就没法获取焦点了
        // need to not have any parents blocking us
        if (hasAncestorThatBlocksDescendantFocus()) {
            return false;
        }

        // 获取焦点的处理逻辑
        handleFocusGainInternal(direction, previouslyFocusedRect);
        return true;
    }


    /**
     * Give this view focus. This will cause
     * {@link #onFocusChanged(boolean, int, android.graphics.Rect)} to be called.
     *
     * Note: this does not check whether this {@link View} should get focus, it just
     * gives it focus no matter what.  It should only be called internally by framework
     * code that knows what it is doing, namely {@link #requestFocus(int, Rect)}.
     *
     * @param direction values are {@link View#FOCUS_UP}, {@link View#FOCUS_DOWN},
     *       {@link View#FOCUS_LEFT} or {@link View#FOCUS_RIGHT}. This is the direction which
     *       focus moved when requestFocus() is called. It may not always
     *       apply, in which case use the default View.FOCUS_DOWN.
     * @param previouslyFocusedRect The rectangle of the view that had focus
     *       prior in this View's coordinate system.
     */
    void handleFocusGainInternal(@FocusRealDirection int direction, Rect previouslyFocusedRect) {
        if (DBG) {
            System.out.println(this + " requestFocus()");
        }

        if ((mPrivateFlags & PFLAG_FOCUSED) == 0) {
            mPrivateFlags |= PFLAG_FOCUSED;

            View oldFocus = (mAttachInfo != null) ? getRootView().findFocus() : null;
```

```
            if (mParent != null) {
                // 父亲告诉旧的焦点view，焦点变更，失去了焦点
                mParent.requestChildFocus(this, this);
            }

            if (mAttachInfo != null) {
                mAttachInfo.mTreeObserver.dispatchOnGlobalFocusChange(oldFocus, this);
            }

            // 这个函数很重要，编辑类view(比如EditText)和普通view的差别就在此
            // 和输入法相关的处理也在此
            onFocusChanged(true, direction, previouslyFocusedRect);
            refreshDrawableState();
        }
    }

    /**
     * Called by the view system when the focus state of this view changes.
     * When the focus change event is caused by directional navigation, direction
     * and previouslyFocusedRect provide insight into where the focus is coming from.
     * When overriding, be sure to call up through to the super class so that
     * the standard focus handling will occur.
     *
     * @param gainFocus True if the View has focus; false otherwise.
     * @param direction The direction focus has moved when requestFocus()
     *                  is called to give this view focus. Values are
     *                  {@link #FOCUS_UP}, {@link #FOCUS_DOWN}, {@link #FOCUS_LEFT},
     *                  {@link #FOCUS_RIGHT}, {@link #FOCUS_FORWARD}, or {@link #FOCUS_BACKWARD}.
     *                  It may not always apply, in which case use the default.
     * @param previouslyFocusedRect The rectangle, in this view's coordinate
     *       system, of the previously focused view.  If applicable, this will be
     *       passed in as finer grained information about where the focus is coming
     *       from (in addition to direction).  Will be <code>null</code> otherwise.
     */
    @CallSuper
    protected void onFocusChanged(boolean gainFocus, @FocusDirection int direction,
            @Nullable Rect previouslyFocusedRect) {
        ...
        InputMethodManager imm = InputMethodManager.peekInstance();
        if (!gainFocus) {
            if (isPressed()) {
                setPressed(false);
            }
            if (imm != null && mAttachInfo != null
                    && mAttachInfo.mHasWindowFocus) {
                imm.focusOut(this);
            }
            onFocusLost();
        } else if (imm != null && mAttachInfo != null
                && mAttachInfo.mHasWindowFocus) {
            // 通知IMMS该view获得了焦点，到此，这后面的逻辑就和上面的window获
            // 得焦点导致view和输入法绑定的逻辑一样了
            imm.focusIn(this);
        }

        invalidate(true);
        ...
    }
```