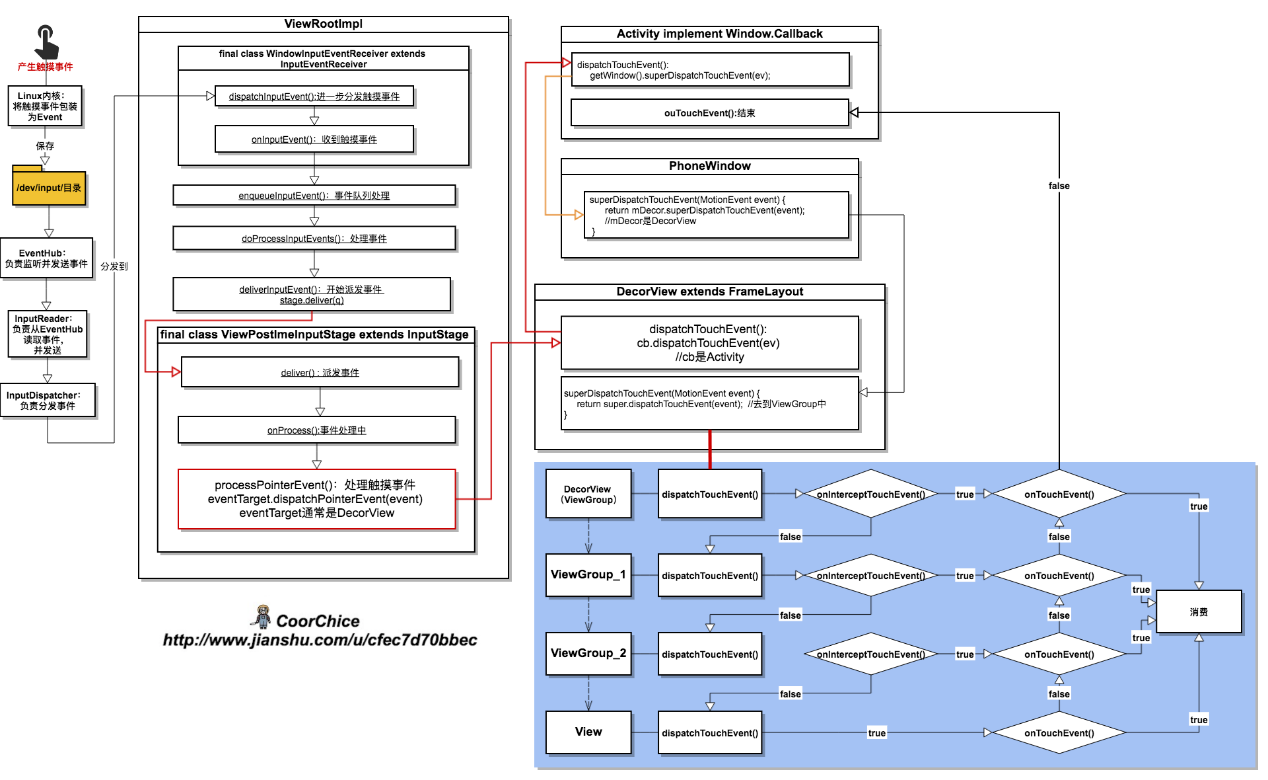
# ViewRootImpl

// frameworks/base/core/java/android/view/ViewRootImpl.java:447



## **WindowInputReceiver**

它继承了**InputEventReceiver**，并且是**ViewRootImpl**的一个内部类

final class WindowInputEventReceiver extends InputEventReceiver｛

@Override

public void onInputEvent(InputEvent event) ｛

...

enqueueInputEvent(event, this, 0, true);

...

｝

｝

当一个输入事件产生时（这里我们认为是触摸事件），会回调**InputEventReceiver.onInputEvent()**。从名字也可以看出，它是接收输入事件的。然后进一步调用 **ViewRootImpl.enqueueInputEvent()** 将输入事件加入单链表队列。

## **enqueueInputEvent**

void enqueueInputEvent(InputEvent event,

InputEventReceiver receiver,

int flags,

boolean processImmediately) {

...

mPendingInputEventTail = q;

//进行队列操作后，

//将ViewRootImpl的mPendingInputEventTail复制为新的触摸事件。

...

if (processImmediately) {

doProcessInputEvents();

//立即处理事件

} else {

scheduleProcessInputEvents();

//走一遍Handler延迟处理事件

}

}

上面这个方法主要对触摸事件进行队列操作(即排了个序)，然后再根据processImmediately参数，决定是立即处理，还是延后处理。

下面看看是如何进行处理的。

void doProcessInputEvents() {

while (mPendingInputEventHead != null) {

...

deliverInputEvent(q);

//进一步派发事件处理

...

}

}

在这个方法中有一个while{}循环体。它会将事件队列循环处理，直到队列中没有数据为

private void deliverInputEvent(QueuedInputEvent q) {

...

InputStage stage;

if (q.shouldSendToSynthesizer()) {

stage = mSyntheticInputStage;

} else {

stage = q.shouldSkipIme() ? mFirstPostImeInputStage : mFirstInputStage;

}

//上面决定将事件派发到那个InputStage中处理

if (stage != null) {

stage.deliver(q);

//派发事件到InputStage中处理

} else {

finishInputEvent(q);

}

}

# **InputStage**

**InputStage** 是在setView()的时候创建的，也就是在**Activity**的onResume()阶段

这些**InputStage**也相当于是单链表结构，一个套一个，比如调用了mFirstPostImeInputStage.deliver(q)。那么SyntheticInputStage, ViewPostImeInputStage, NativePostImeInputStage, EarlyPostImeInputStage都将能够处理这个触摸事件。这里我们主要看看ViewPostImeInputStage是如何处理的。

CharSequence counterSuffix = attrs.getTitle();  
mSyntheticInputStage = **new** SyntheticInputStage();  
InputStage viewPostImeStage = **new** ViewPostImeInputStage(mSyntheticInputStage);  
InputStage nativePostImeStage = **new** NativePostImeInputStage(viewPostImeStage,  
 **"aq:native-post-ime:"** + counterSuffix);  
InputStage earlyPostImeStage = **new** EarlyPostImeInputStage(nativePostImeStage);  
InputStage imeStage = **new** ImeInputStage(earlyPostImeStage,  
 **"aq:ime:"** + counterSuffix);  
InputStage viewPreImeStage = **new** ViewPreImeInputStage(imeStage);  
InputStage nativePreImeStage = **new** NativePreImeInputStage(viewPreImeStage,  
 **"aq:native-pre-ime:"** + counterSuffix);  
  
mFirstInputStage = nativePreImeStage;  
mFirstPostImeInputStage = earlyPostImeStage;

SyntheticInputStage<- viewPostImeStage<- nativePostImeStage <- **earlyPostImeStage** <- imeStage <- viewPreImeStage <- nativePreImeStage

## ViewPostImeInputStage

 合成的。

## setView

http://www.jcodecraeer.com/a/anzhuokaifa/androidkaifa/2016/0302/4025.html

<https://www.jianshu.com/p/b7cef3b3e703>

# WindowManagerPolicy

frameworks\native\include\input\input.h

frameworks/base/core/java/android/view/WindowManagerPolicy.java.

|  |  |  |
| --- | --- | --- |
|  |  |  |
| POLICY\_FLAG\_WAKE | 该事件应该唤醒设备 |  |
| POLICY\_FLAG\_VIRTUAL | 虚拟按键：电容处理  // Indicates that the key is virtual, such as a capacitive button, and should  // generate haptic feedback. Virtual keys may be suppressed for some time  // after a recent touch to prevent accidental activation of virtual keys adjacent  // to the touch screen during an edge swipe. |  |
| POLICY\_FLAG\_FUNCTION | Indicates that the key is the special function modifier. |  |
| POLICY\_FLAG\_GESTURE = 0x00000008, | special gesture that has been detected by  // the touch firmware or driver. Causes touch events from the same device to be canceled. |  |
| POLICY\_FLAG\_RAW\_MASK = 0x0000ffff, |  |  |
| POLICY\_FLAG\_INJECTED | input event was injected. |  |
| POLICY\_FLAG\_TRUSTED  0x02000000, | input event is from a trusted source such as a directly attached  // input device or an application with system-wide event injection permission. | 默认链接的host的都会有这个值，可信输入 |
| POLICY\_FLAG\_FILTERED | event has passed through an input filter. |  |
| POLICY\_FLAG\_DISABLE\_KEY\_REPEAT |  |  |
| POLICY\_FLAG\_INTERACTIVE  0x20000000 | /\* These flags are set by the input reader policy as it intercepts each event. \*/  // Indicates that the device was in an interactive state when the  // event was intercepted. | 可交互的，亮屏就有 |
| POLICY\_FLAG\_PASS\_TO\_USER | // Indicates that the event should be dispatched to applications.  // The input event should still be sent to the InputDispatcher so that it can see all  // input events received include those that it will not deliver. |  |

:

/\*\*

\* Pass this event to the user / app. To be returned from

\* {@link #interceptKeyBeforeQueueing}.

\*/

public final static int ACTION\_PASS\_TO\_USER = 0x00000001;

ACTION\_PASS\_TO\_USER