


原

Android 5.0(Lollipop)事件输入系统(Input System)

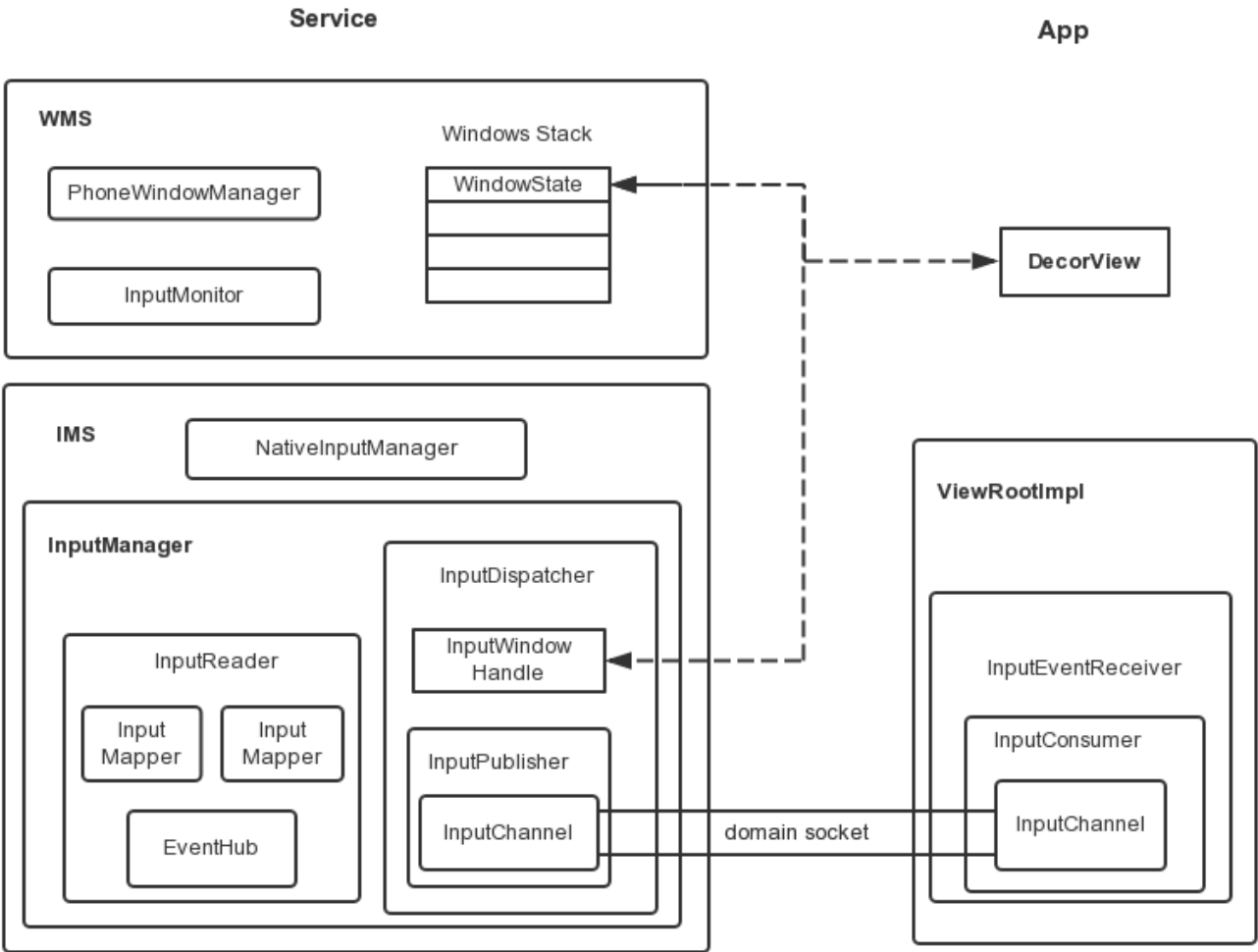
2014年12月13日 22:12:50

ariesjzj

阅读数：12555

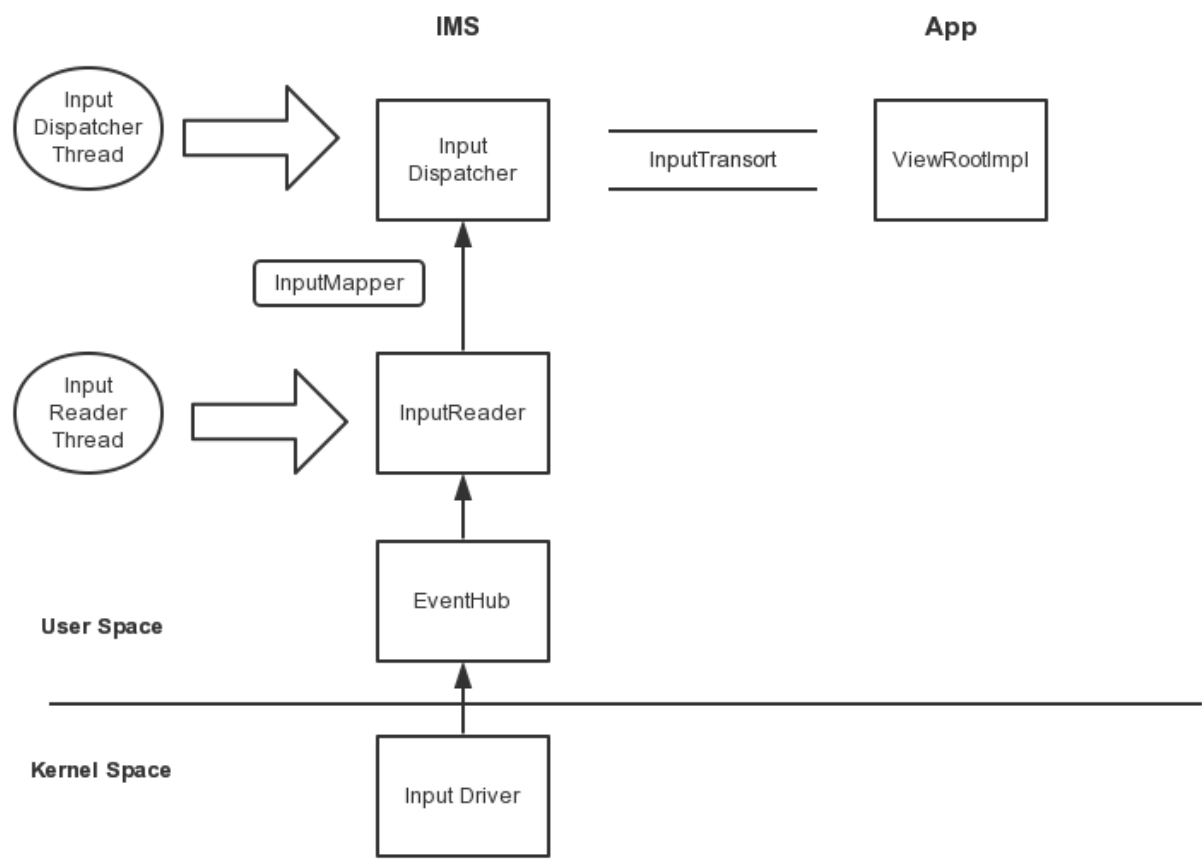
 版权声明：本文为博主原创文章，只要标明出处即可转载。 <https://blog.csdn.net/ariesjzj/article/details/41909159>

其实Android 5.0中事件输入子系统的框架和流程没有本质变化。Service端的实现在/frameworks/native/services/inputflinger/下（4.4中在/frameworks/base/services/java/com/android/server/input/下）。通用部分的实现在/frameworks/native/libs/input/下。Android系统中负责管理输入事件的主要是InputManagerService（IMS）。它主要的任务就是从设备中读事件数据，然后将输入事件分发到各个窗口中去，另外还需要让系统有机会来处理一些系统按键。显然，要完成这个工作，IMS需要与其它模块打交道，其中最主要的就是WMS和ViewRootImpl。主要的几个模块如下：



<http://blog.csdn.net/jinzhuojun>

WindowManagerService(WMS)是窗口管理服务，核心维护了一个有序的窗口堆栈。PhoneWindowManager(PWM)里有关于手机策略的实现，和输入相关的主要是对系统按键的处理。InputManagerService是输入管理服务，主要干活的是Native层的InputManager。InputManager中的InputReader负责使用EventHub从Input driver中拿事件，然后让InputMapper解析。接着传给InputDispatcher，InputDispatcher负责一方面将事件通过InputManager，InputMonitor一路传给PhoneWindowManager来做系统输入事件的处理，另一方面将这些事件传给焦点及监视窗口。NativeInputManager实现InputReaderPolicyInterface和InputDispatcherPolicyInterface接口，在Native层的InputManager和Java层的IMS间起到一个胶水层的作用。InputMonitor实现了WindowManagerCallbacks接口，起到了IMS到WMS的连接作用。App这边，ViewRootImpl相当于App端一个顶层View的Controller。这个顶层View在WMS中对应一个窗口，用WindowState描述。WindowState中有InputWindowHandle代表一个接收输入事件的窗口句柄。InputDispatcher中的mFocusedWindowHandle指示了焦点窗口的句柄。InputDispatcher管理了一坨连接（一个连接对应一个注册到WMS的窗口），通过这些个连接InputDispatcher可以直接将输入事件发往App端的焦点窗口。输入事件从Driver开始的处理过程大致如下：

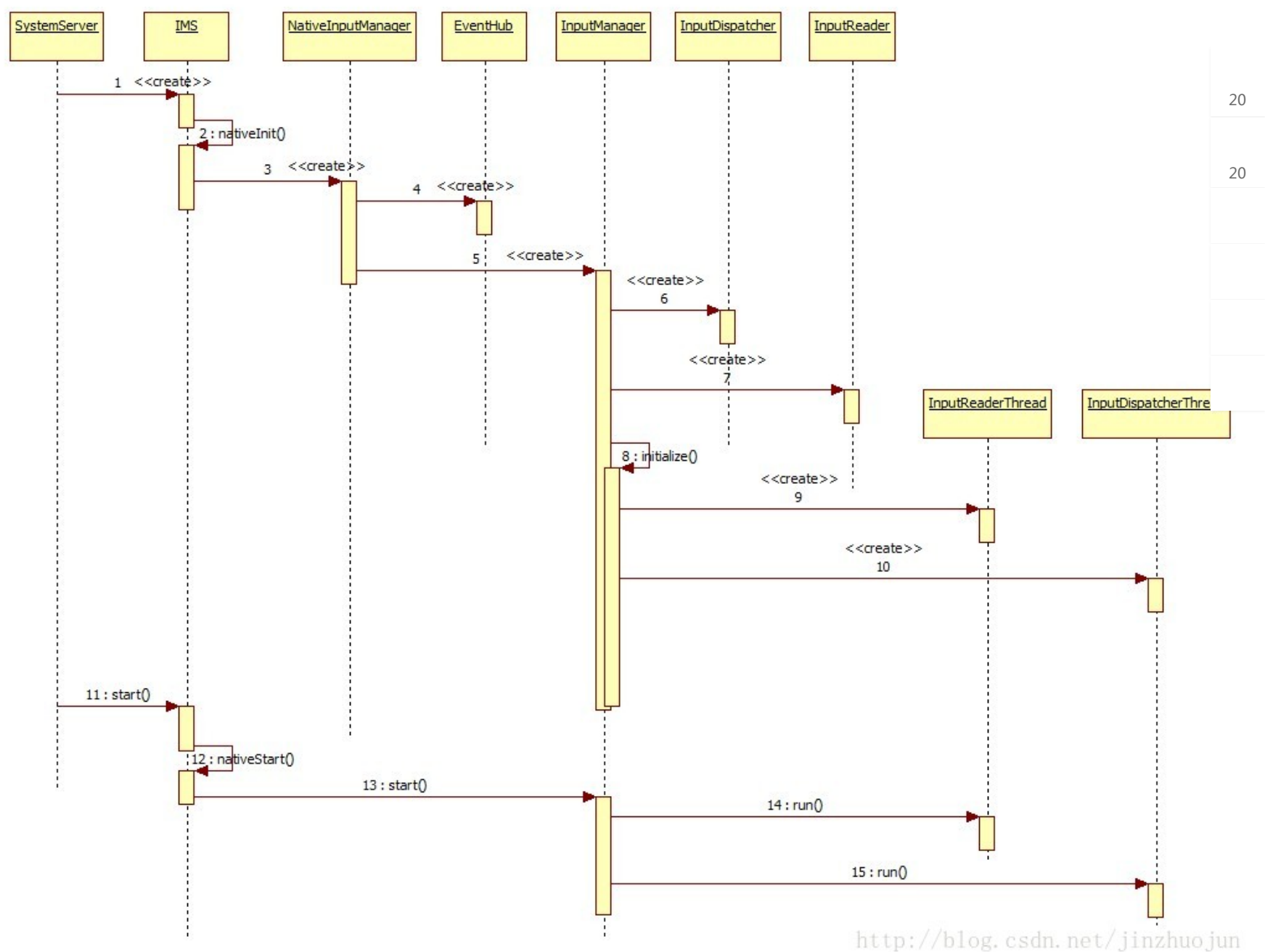


<http://blog.csdn.net/jinzhuojun>

为了故事的完整性，还是先看下初始化。SystemServer中初始化IMS，然后初始化WMS，把IMS作为参数传入。

```
1 470      Slog.i(TAG, "Input Manager");
2 471      inputManager = new InputManagerService(context);
3 472
4 473      Slog.i(TAG, "Window Manager");
5 474      wm = WindowManagerService.main(context, inputManager,
6 475          mFactoryTestMode != FactoryTest.FACTORY_TEST_LOW_LEVEL,
7 476          !mFirstBoot, mOnlyCore);
8 ...
9 483      inputManager.start();
```

IMS的构造函数中，调用nativeInit()来初始化。注意这里拿了DisplayThread的Handler，意味着IMS中的消息队列处理都是在单独的DisplayThread中进行的。它是系统中共享的单例前台线程，主要用作输入输出的处理用。这样可以用户体验敏感的处理少受其它工作的影响，减少延时。整个初始化过程流程如下：

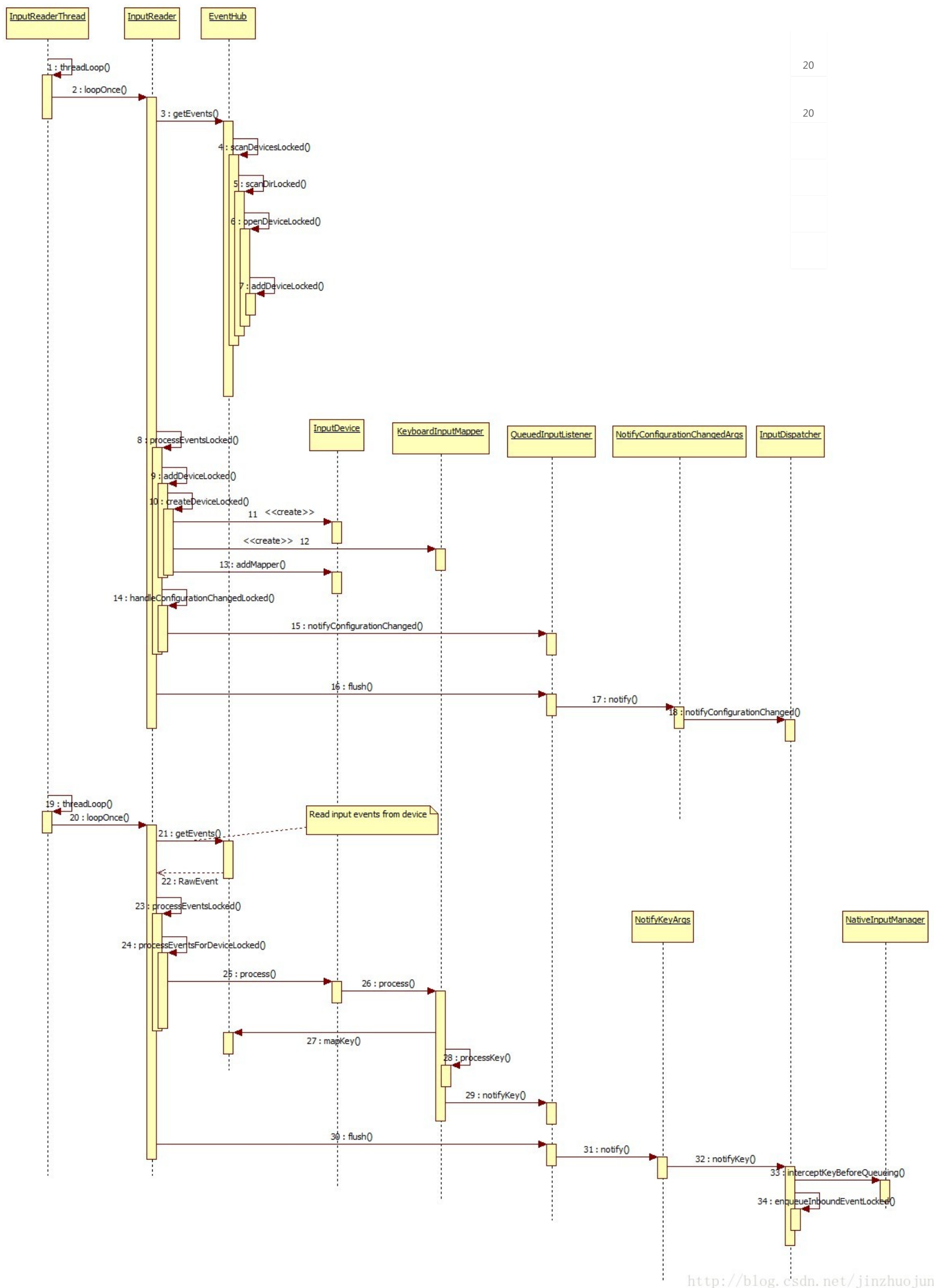


可以看到，初始化时依次初始化NativeInputManager，EventHub，InputManager，InputDispatcher，InputReader，InputReaderThread，InputDispatcherThread。NativeInputManager可看作IMS和InputManager的中间层，将IMS的请求转化为对InputManager及其内部对象的操作，同时将InputManager中模块的请求通过JNI调回IMS。InputManager是输入控制中心，它有两个关键线程InputReaderThread和InputDispatcherThread，它们的主要功能部分分别在InputReader和InputDispatcher。前者用于从设备中读取事件，后者将事件分发给目标窗口。EventHub是输入设备的控制中心，它直接与input driver打交道。负责处理输入设备的增减，查询，输入事件的处理并向上层提供getEvents()接口接收事件。在它的构造函数中，主要做三件事：

1. 创建epoll对象，之后就可以把各输入设备的fd挂上面多路等待输入事件。
2. 建立用于唤醒的pipe，把读端挂到epoll上，以后如果有设备参数的变化需要处理，而getEvents()又阻塞在设备上，就可以调用wake()在pipe的写端写入，就可以让线程从等待中返回。
3. 利用inotify机制监听/dev/input目录下的变更，如有则意味着设备的变化，需要处理。

因为事件的处理是流水线，需要InputReader先读事件，然后InputDispatcher才能进一步处理和分发。因此InputDispatcher需要监听InputReader。这里使用了Listener模式，InputDispatcher作为InputReader构造函数的第三个参数，它实现InputListenerInterface接口。到了InputReader的构造函数中，将之包装成QueuedInputListener。QueuedInputListener中的成员变量mArgsQueue是一个缓冲队列，只有在flush()时，才会一次性通知InputDispatcher。QueuedInputListener应用了Command模式，它通过包装InputDispatcher(实现InputListenerInterface接口)，将事件的处理请求封装成NotifyArgs，使其有了缓冲执行的功能。

全初始化好后，SystemServer调用start()函数让InputManager中两个线程开始运行。先看InputReaderThread，它是事件在用户态处理过程的起点。这里以按键事件的处理为例。



InputReaderThread不断调用InputReader的pollOnce()->getEvents()函数来得到事件，这些事件可以是输入事件，也可以是由inotify监测到设备增减变更所触发的事件。第一次进入时会扫描/dev/input目录建立设备列表，存在mDevice成员变量中(EventHub中有设备列表KeyedVector<int32\_t, Device\*> mDevices；对应的，InputReader中也有设备列表KevedVector<int32 t, InputDevice\*> mDevices。这里先添加到前者，然后会在InputReader::addDeviceLocked()中添加到后者。)，

变，会根据实际情况调用addDeviceLocked(), removeDeviceLocked()和handleConfigurationChangedLocked()。对于其它设备中来的输入事件，会调用processEventsForDeviceLocked()进一步处理。其中会根据当时注册的InputMapper对事件进行处理，然后将事件处理请求放入缓冲队列（ QueuedInputListener中的mArgsQueue ）。

```
1 155void QueuedInputListener::notifyKey(const NotifyKeyArgs* args) {
2 156     mArgsQueue.push(new NotifyKeyArgs(*args));
3 157}
```

InputMapper是做什么的呢，它是用于解析原始输入事件的。比如back, home等VirtualKey，传上来时是个Touch事件，这里要根据坐标转化为相应的按键事件。多点触摸时，需要计算每个触摸点分别属于哪条轨迹，这本质上是个二分图匹配问题，这也是在InputMapper中完成的。回到流程主线上，在InputReader的loopOnce()的结尾会调用QueuedInputListener::flush()统一回调缓冲队列中各元素的notify()接口：

```
1 171void QueuedInputListener::flush() {
2 172     size_t count = mArgsQueue.size();
3 173     for (size_t i = 0; i < count; i++) {
4 174         NotifyArgs* args = mArgsQueue[i];
5 175         args->notify(mInnerListener);
6 176         delete args;
7 177     }
8 178     mArgsQueue.clear();
9 179}
```

以按键事件为例，最后会调用到InputDispatcher的notifyKey()函数中。这里先将参数封装成KeyEvent：

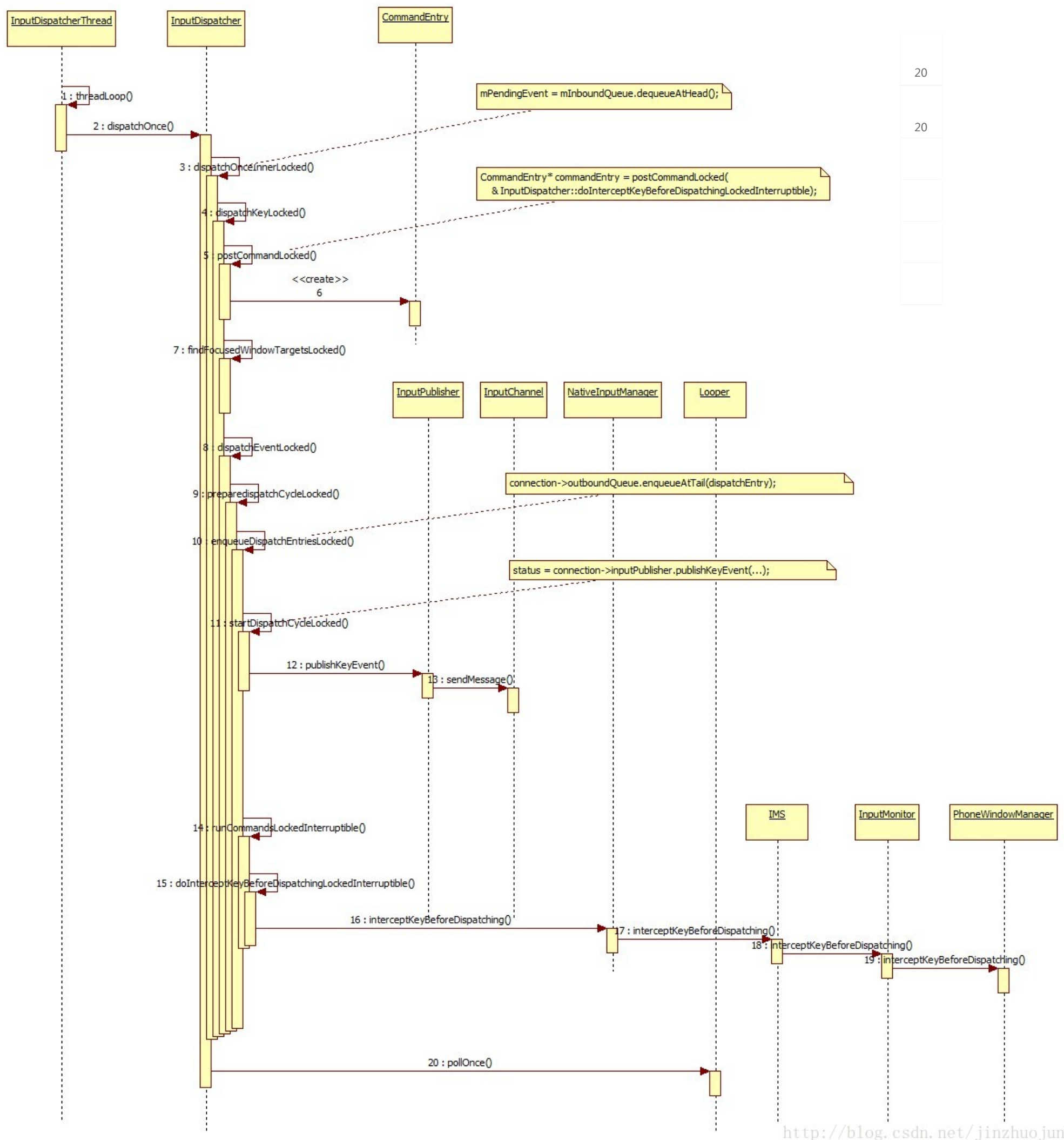
```
1 2416     KeyEvent event;
2 2417     event.initialize(args->deviceId, args->source, args->action,
3 2418                    flags, keyCode, args->scanCode, metaState, 0,
4 2419                    args->downTime, args->eventTime);
```

然后把它作为参数调用NativeInputManager的interceptKeyBeforeQueueing()函数。顾名思义，就是在放到待处理队列前看看是不是需要系统处理的系统按键，它会通过JNI调回Java世界，最终调到PhoneWindowManager的interceptKeyBeforeQueueing()。然后，基于输入事件信息创建KeyEntry对象，调用enqueueInboundEventLocked()将之放入队列等待InputDiaptcherThread线程拿出处理。

```
1 2439         KeyEntry* newEntry = new KeyEntry(args->eventTime,
2 2440                                           args->deviceId, args->source, policyFlags,
3 2441                                           args->action, flags, keyCode, args->scanCode,
4 2442                                           metaState, repeatCount, args->downTime);
5 2443
6 2444         needWake = enqueueInboundEventLocked(newEntry);
```

下面该InputDispatcherThread登场了。





<http://blog.csdn.net/jinzhuojun>

可以看到，InputDisptacher的主要任务是前面收到的输入事件发送到PWM及App端的焦点窗口。前面提到InputReaderThread中收到事件后会调用notifyKey()来通知InputDispatcher，也就是放在mInboundQueue中，在InputDispatcher的dispatchOnce()函数中，会从这个队列拿出处理。

```

1 234     if (!haveCommandsLocked()) {
2 235         dispatchOnceInnerLocked(&nextWakeupTime);
3 236     }
4 ...
5 240     if (runCommandsLockedInterruptible()) {
6 241         nextWakeupTime = LONG_LONG_MIN;
7 242     }
  
```

其中dispatchOnceInnerLocked()会根据拿出的EventEntry类型调用相应的处理函数，以Key事件为例会调用dispatchKeyLocked()：

```

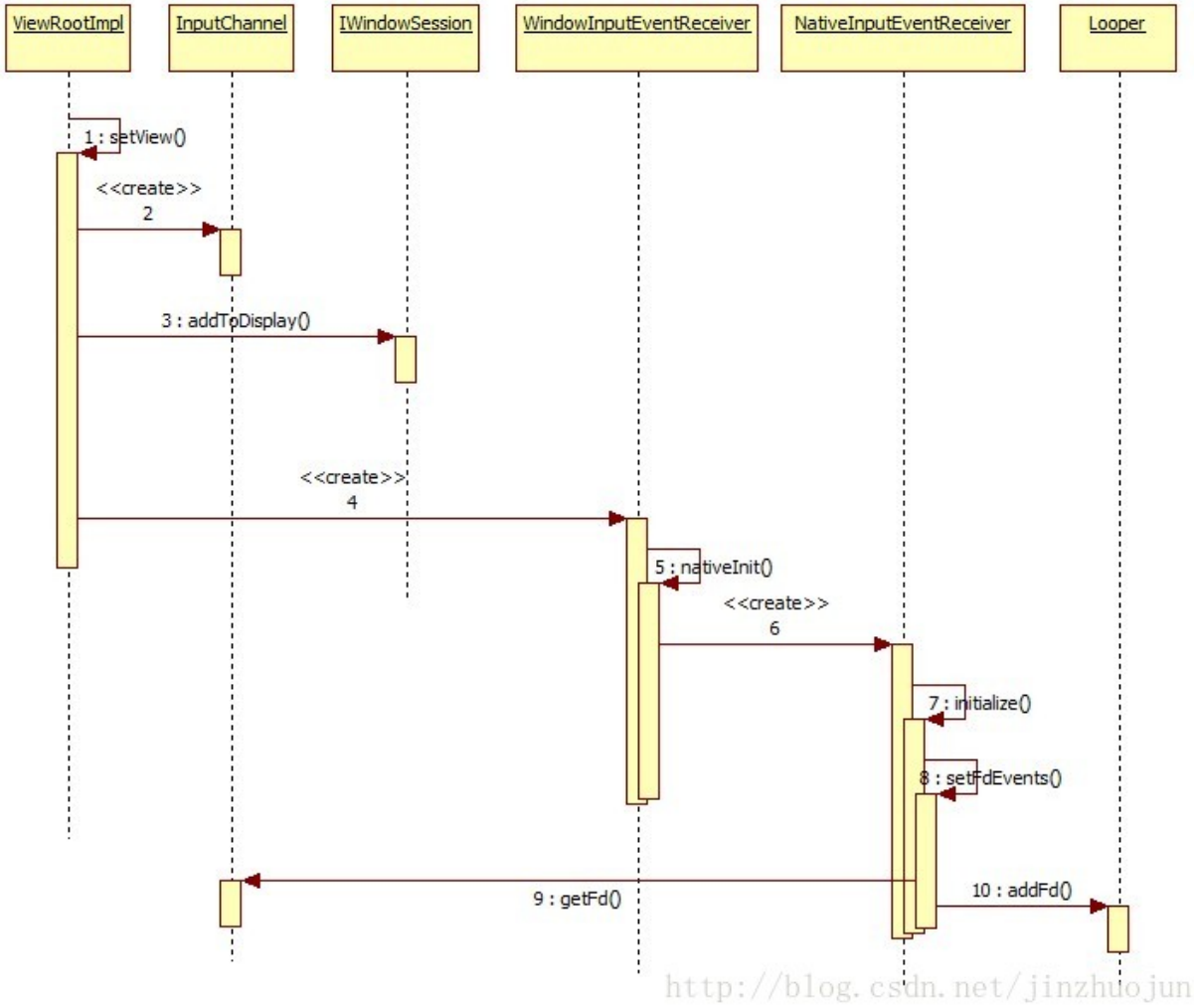
1 767     CommandEntry* commandEntry = postCommandLocked(
2 768         & InputDispatcher::doInterceptKeyBeforeDispatchingLockedInterruptible);
3 769     if (mFocusedWindowHandle != NULL) {
  
```

```
7 | ...
8 | 791 // Identify targets.
9 | 792 Vector<InputTarget> inputTargets;
10| 793 int32_t injectionResult = findFocusedWindowTargetsLocked(currentTime,
11| 794     entry, inputTargets, nextWakeupTime);
12| ..
13| 804 addMonitoringTargetsLocked(inputTargets);
14| 805
15| 806 // Dispatch the key.
16| 807 dispatchEventLocked(currentTime, entry, inputTargets);
```

20
20

它会找到目标窗口，然后通过之前和App间建立连接发送事件。如果是个需要系统处理的Key事件，这里会封装成CommandEntry插入到mCommandQueue队列中的runCommandLockedInterruptible()函数中会调用doInterceptKeyBeforeDispatchingLockedInterruptible()来让PWM有机会进行处理。最后dispatchOnce()调用pollC...人和App的连接上接收处理完成消息。那么，InputDispatcher是怎么确定要往哪个窗口中发事件呢？这里的成员变量mFocusedWindowHandle指示了焦点窗口，然后findFocusedWindowLocked()会调用一系列函数（handleTargetsNotReadyLocked(), checkInjectionPermission(), checkWindowReadyForMoreInputLocked()等）检查mFocusedWindowHandle是否能接收输入。如果可以，将之以InputTarget的形式加到目标窗口数组中。然后就会调用dispatchEventLocked()进行发送。那么，这个mFocusedWindowHandle是如何维护的呢？为了更好地理解，这里回头分析下窗口连接的管理及焦点窗口的管理。

在App端，新的顶层窗口需要被注册到WMS中，这是在ViewRootImpl::setView()中做的。



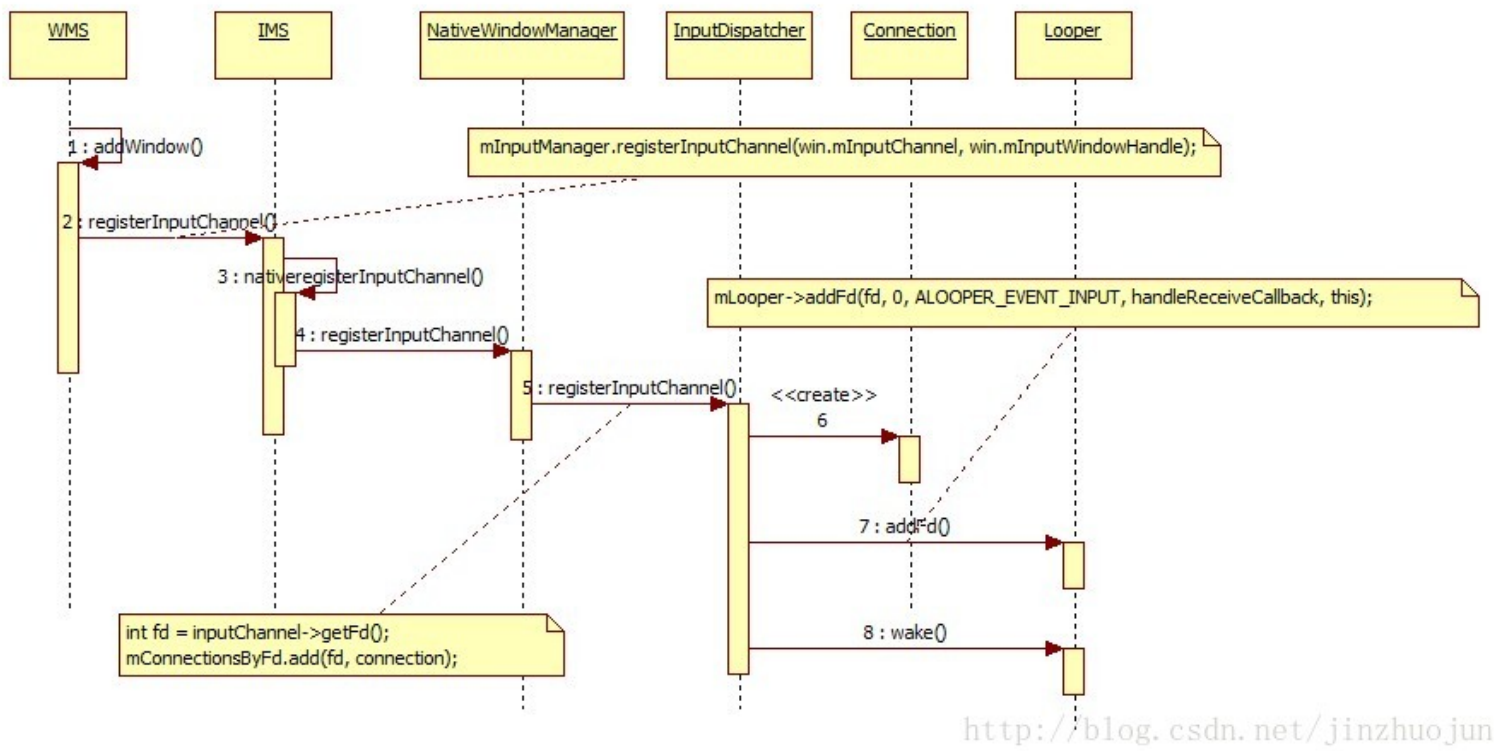
其中与输入相关的主要有以下几步：  
先创建InputChannel，注意还没初始化。

```
521 mInputChannel = new InputChannel();
```

初始化InputChannel，通过WMS的相应接口addToDisplay()：

```
1 | 527 res = mWindowSession.addToDisplay(mWindow, mSeq, mWindowAttributes,
2 | 528     getHostVisibility(), mDisplay.getDisplayId(),
3 | 529     mAttachInfo.mContentInsets, mInputChannel);
```

WMS会建立与InputDispatcher的连接。流程如下：



20
20

ViewRootImpl通过Session中的addToDisplay()会最终调用WMS的addWindow()。在WMS中，会创建一对InputChannel，本质上是一对本地socket。然后一个注册给InputDispatcher，一个作为输出参数传回给App的ViewRootImpl。这样就建立了App与IMS的一对连接。

```
1 2409         if (outInputChannel != null && (attrs.inputFeatures
2 2410             & WindowManager.LayoutParams.INPUT_FEATURE_NO_INPUT_CHANNEL) == 0) {
3 2411             String name = win.makeInputChannelName();
4 2412             InputChannel[] inputChannels = InputChannel.openInputChannelPair(name);
5 2413             win.setInputChannel(inputChannels[0]);
6 2414             inputChannels[1].transferTo(outInputChannel);
7 2415
8 2416             mInputManager.registerInputChannel(win.mInputChannel, win.mInputWindowHandle);
9 2417         }
```

在InputDispatcher::registerInputChannel()中：

```
1 3327         sp<Connection> connection = new Connection(inputChannel, inputWindowHandle, monitor);
2 3328
3 3329         int fd = inputChannel->getFd();
4 3330         mConnectionsByFd.add(fd, connection);
5 ...
6 3336         mLooper->addFd(fd, 0, ALOOPER_EVENT_INPUT, handleReceiveCallback, this);
```

这里创建的Connection表示一个InputDispatcher到应用窗口的连接，里边除了用于传输的inputChannel，inputPublisher和表示事件接收窗口的inputWindowHandle，还有两个队列，outboundQueue是要发的事件，waitQueue是已发事件但还没有从App端收到完成通知的。这是因为对于一些事件，Input Dispatcher在App没处理完前一个时不会发第二个。mLooper->addFd()将相应的fd放入InputDispatcher等待的集合中，回调函数为handleReceiveCallback()，也就是说InputDispatcher在收到App发来的消息时是调用它进行处理的。最后调用mLooper->wake()使InputDispatcherThread从epoll\_wait()中返回。

回到App端，如果前面没啥问题，接下来会创建WindowInputEventReceiver，它是App的事件接收端。

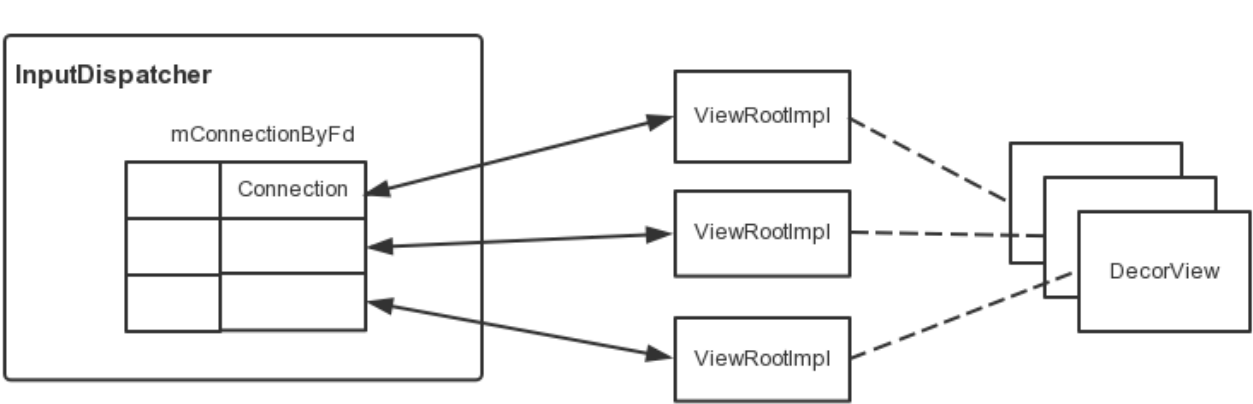
```
1 607         mInputEventReceiver = new WindowInputEventReceiver(mInputChannel,
2 608             Looper.myLooper());
```

初始化完后，这个连接的fd就被挂到主线程的等待fd集合去了(InputEventReceiver::nativeInit())。也就是说，当连接上有消息来，主线程就会调用相应的回调处理NativeInputEventReceiver::handleEvent()。

接下来初始化App端事件处理的流水线，这里使用了Chain of responsibility模式，让事件经过各个InputStage，每一个Stage可以决定是否自己处理，也可以传递给下一家。下一家也是如此。在后面的handleEvent()可以看到它们的用法。

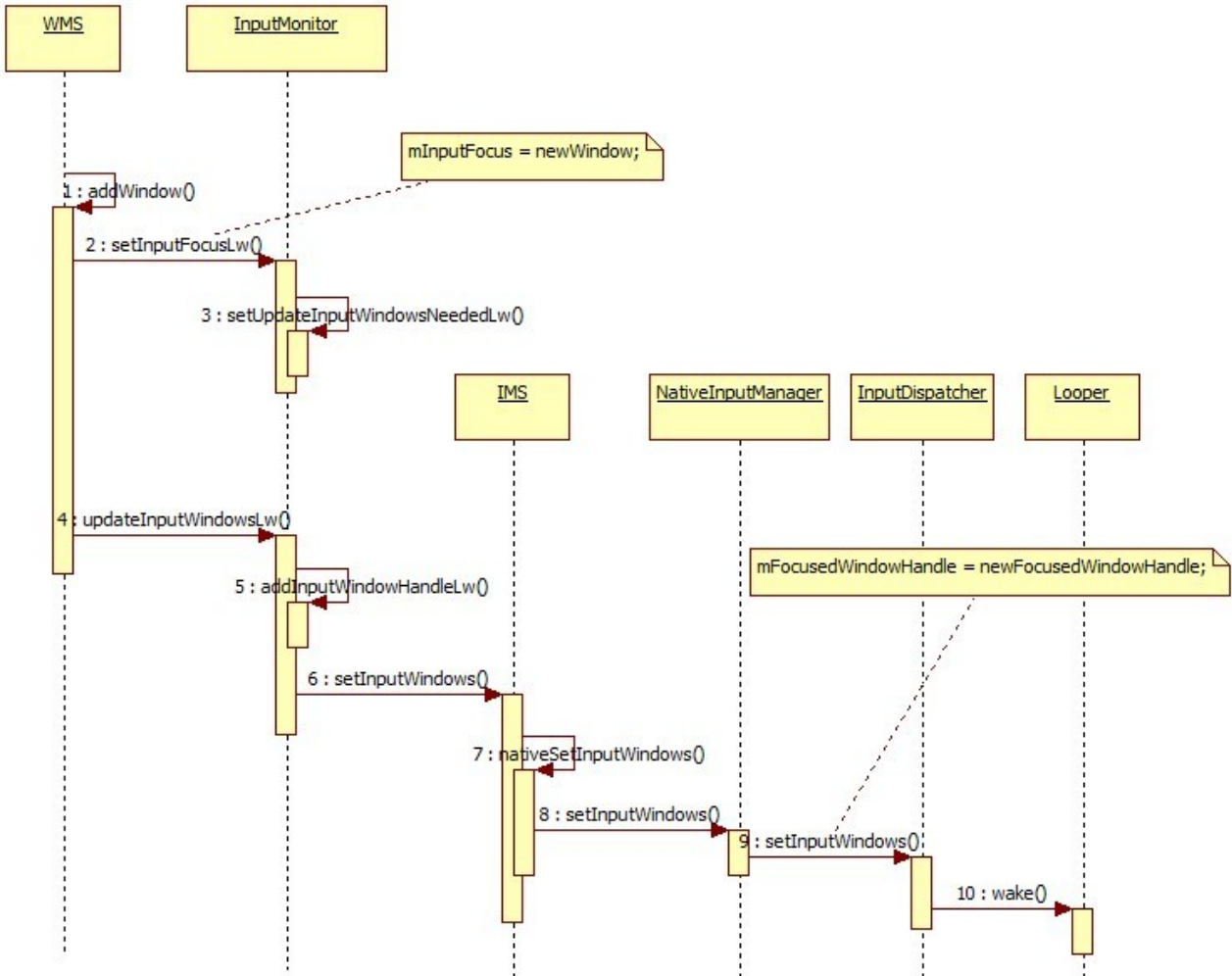
```
1 623         // Set up the input pipeline.
2 624         CharSequence counterSuffix = attrs.getTitle();
3 625         mSyntheticInputStage = new SyntheticInputStage();
4 626         InputStage viewPostImeStage = new ViewPostImeInputStage(mSyntheticInputStage);
5 627         InputStage nativePostImeStage = new NativePostImeInputStage(viewPostImeStage,
6 628             "aq:native-post-ime:" + counterSuffix);
7 629         InputStage earlyPostImeStage = new EarlyPostImeInputStage(nativePostImeStage);
8 630         InputStage imeStage = new ImeInputStage(earlyPostImeStage,
9 631             "aq:ime:" + counterSuffix);
10 632         InputStage viewPreImeStage = new ViewPreImeInputStage(imeStage);
11 633         InputStage nativePreImeStage = new NativePreImeInputStage(viewPreImeStage,
12 634             "aq:native-pre-ime:" + counterSuffix);
13 635
14 636         mFirstInputStage = nativePreImeStage;
15 637         mFirstPostImeInputStage = earlyPostImeStage;
```





<http://blog.csdn.net/jinzhuojun>

连接建立后，接下来要考虑WMS如何将焦点窗口信息传给InputDispatcher。举例来说，当新的窗口加入到WMS中，一般焦点会放到新加窗口上。来看下WMS中的addWindow()函数。



<http://blog.csdn.net/jinzhuojun>

首先，当焦点需要变化时。当焦点窗口变化时，WMS调用

```
mInputMonitor.setInputFocusLw(mCurrentFocus, updateInputWindows);
```

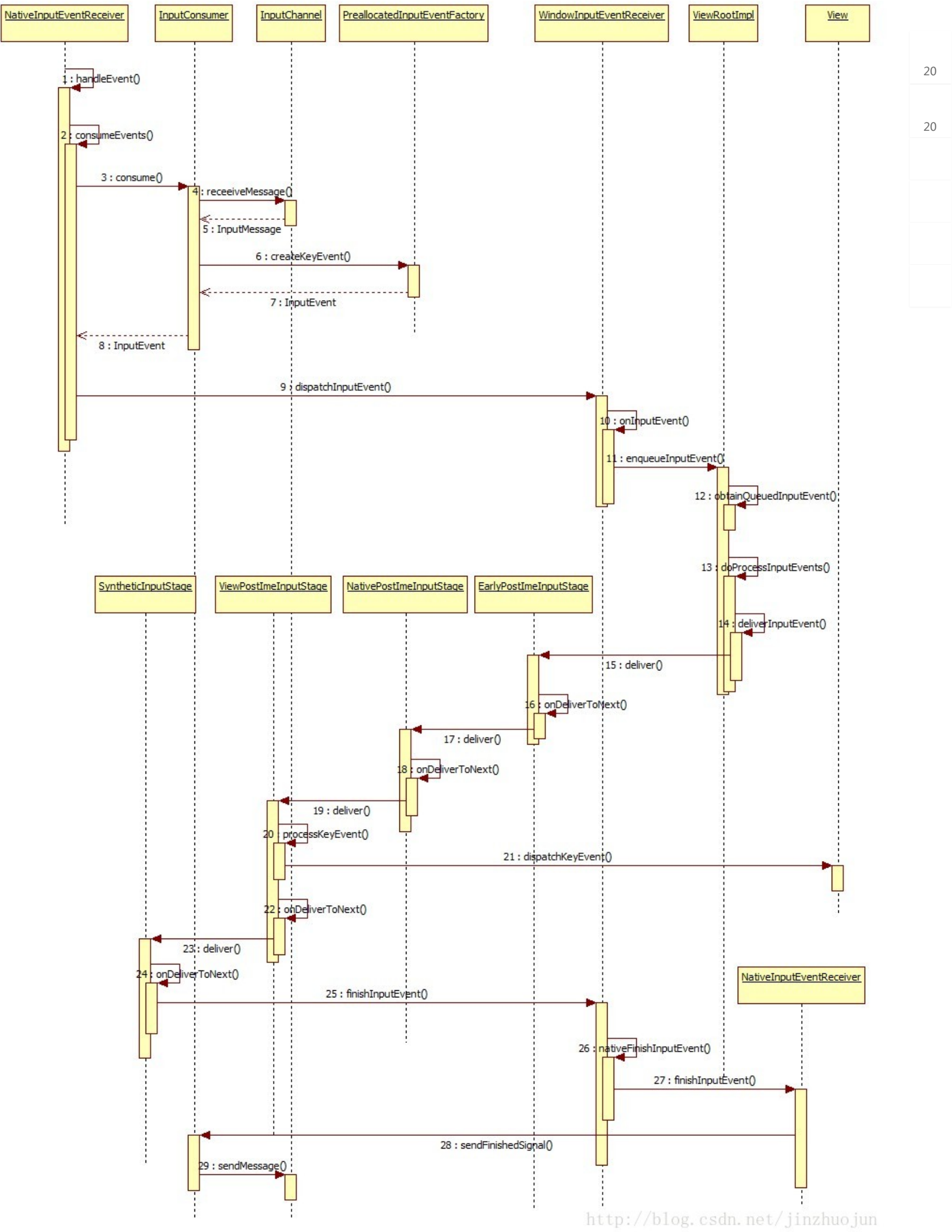
将焦点窗口设到InputMonitor的mInputFocus中。然后调用

```
mInputMonitor.updateInputWindowsLw(true);
```

来创建InputWindowHandle列表，其中被设成焦点窗口的InputWindowHandle的hasFocus会被置位。之后会调用

```
mService.mInputManager.setInputWindows(mInputWindowHandles);
```

将这些信息传到Native层的InputDispatcher。这样InputDispatcher就能够知道要往哪个窗口传事件。在InputDispatcher的setInputWindows()中，会更新InputDispatcher中的焦点窗口句柄。这样，InputDispatcher中就记录下了焦点窗口信息。当IMS的InputDispatcher通过InputChannel发事件到焦点窗口时，NativeInputEventReceiver的handleEvent()会被调用。



<http://blog.csdn.net/jinzhuojun>

基本流程比较直观，先接收事件，然后放入ViewRootImpl的处理队列，然后dispatch给View处理，经过上面提到的一系列InputStage，最后App处理完事件后还需要向IMS发送一个完成信号。

注意上面是以Key事件为例的。对于Motion事件就有差别了。因为触摸移动中的事件不一定要每一个都处理，因为显示也就60HZ，你如果100HZ的输入事件，全处理只会浪费计算资源。上面这条路是每当InputDispatcher有事件发过来时就会触发的，而对于Motion事件，系统会把一个VSync周期内的事件存为Batch，当VSync到来时一起处理。从JB开始，App对输入事件的处理是由VSync信号来驱动的。可以看Choreographer中的VSync回调中首先处理的就是输入事件。

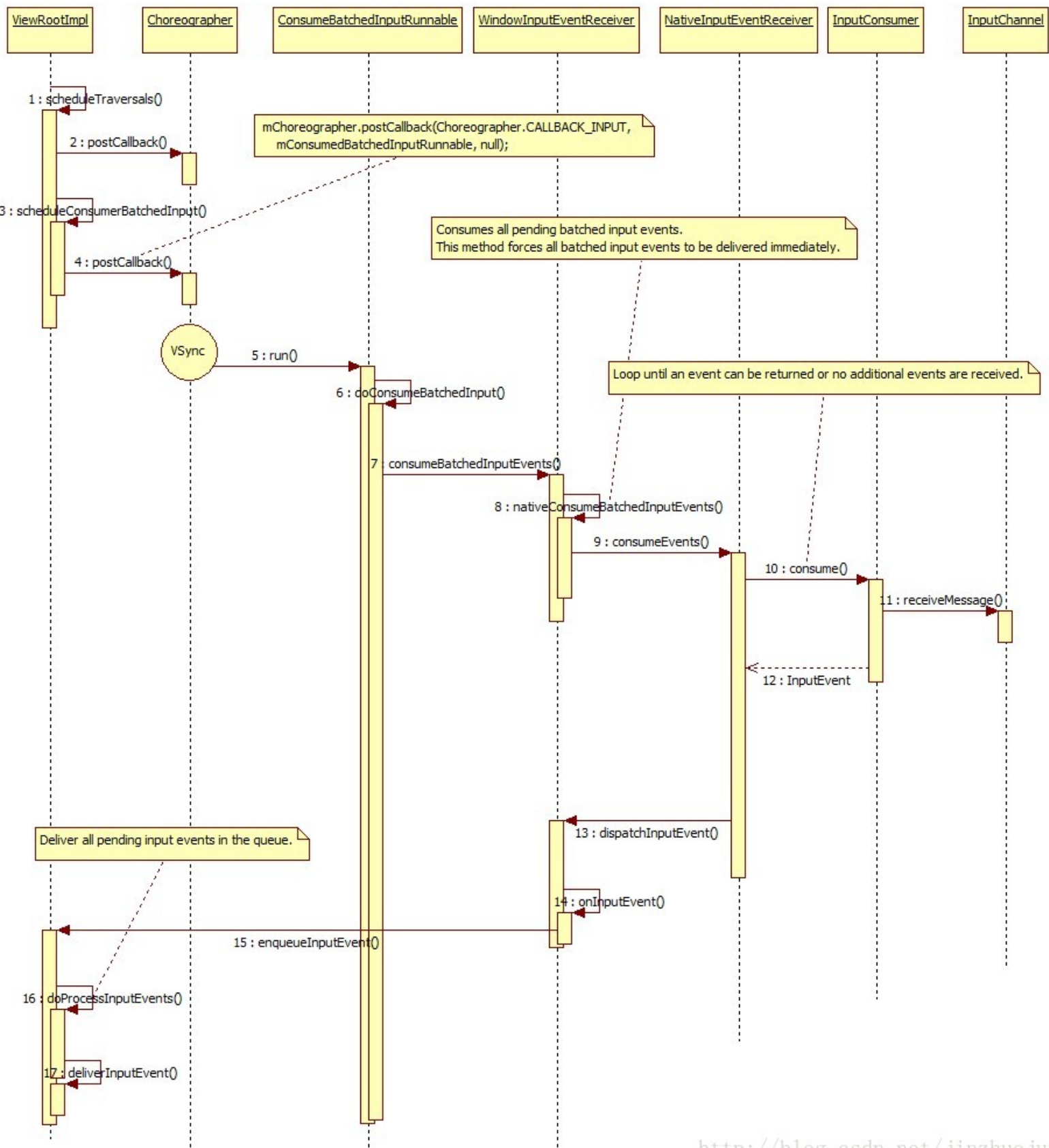
```
1 | 542 | doCallbacks(Choreographer.CALLBACK_INPUT, frameTimeNanos);
2 | 543 | doCallbacks(Choreographer.CALLBACK_ANIMATION, frameTimeNanos);
3 | 544 | doCallbacks(Choreographer.CALLBACK_TRAVERSAL, frameTimeNanos);
```

在非VSync触发的情况下，NativeInputEventReceiver::handleEvent()调用consumeEvents()时参数consumeBatches为false，通过InputConsumer::consume()函数得知，它会被放到Batch当中：

循环退出条件不满足所以一直读到receiveMessage()失败，退出后在consumeEvents()中由于返回值不为0所以事件不会被马上处理；而当VSync信号到来时，下一个consumeEvents()的参数consumeBatches为true，意味着要处理Batch。

20

20



<http://blog.csdn.net/jinzhuojun>

ViewRootImpl中维护了pending input event的列表，用mPendingInputEventHead和mPendingInputEventTail指示，其中的元素为QueuedInputEvent类型。这个队列会由InputEventReceiver调用enqueueInputEvent()加入元素，然后ViewRootImpl延时或非延时在doProcessInputEvents()中读出并处理。处理的方法如前图调用deliverInputEvent()，然后调用InputStage的deliver()方法进行处理。最后调用finishInputEvent()向IMS发送完成信息，它实际是调用了NativeInputEventReceiver::finishInputEvent()，该函数内部使用InputConsumer的sendFinishedSignal()发送处理结束信号。

事实上，当Input resample机制打开时（系统属性ro\_input.noresample控制），对于Motion事件在InputConsumer里的处理会更复杂一点。Android在InputConsumer中加入了Motion事件的重采样。VSync代表Display已完成一帧的显示，系统需要准备下一帧，也就是说我们需要知道VSync信号来到时触摸的坐标。那么问题来了，输入设备的事件不是按VSync来的，比如是10ms为周期（VSync周期为16.67ms）。那么怎么知道VSync的时刻的输入坐标呢，只能靠估计。怎么估计呢，靠的是采样加上内外插值结合的估值方法。详细可参见<http://www.masonchang.com/blog/2014/8/25/androids-touch-resampling-algorithm> 和 <http://developer.sonymobile.com/2014/09/30/understanding-touch-resampling-in-smartphones-touchscreen-technology-series-3/>。

前面提到，InputDispatcher除了给App端传事件外，还有个任务是处理系统按键。系统中有一些特殊的按键，是系统需要处理的，如音量，电源键等。它是通过interceptKeyBeforeDispatching()和interceptKeyBeforeQueueing()两个函数来截获的。interceptKeyBeforeDispatching()主要用于处理home, menu和search，interceptKeyBeforeQueueing()主要用于处理音量和电源键。这些处理的实现都放在PWM中，而调用者是在InputDispatcherThread。这样的好处是把平台相关的东西都放在PWM中，而InputDispatcher中是平台通用的东西，厂商要定制策略只需修改PWM。这样，做到了Mechanism和Policy的分离。那InputDisaptcher是如何调用到PWM中的呢？首先InputDispatcher中的代码中有几个hook的地方：

```
1 | 3510     nsecs_t delay = mPolicy->interceptKeyBeforeDispatching(commandEntry->inputWindowHandle,
2 | 3511         &event, entry->policyFlags);
```

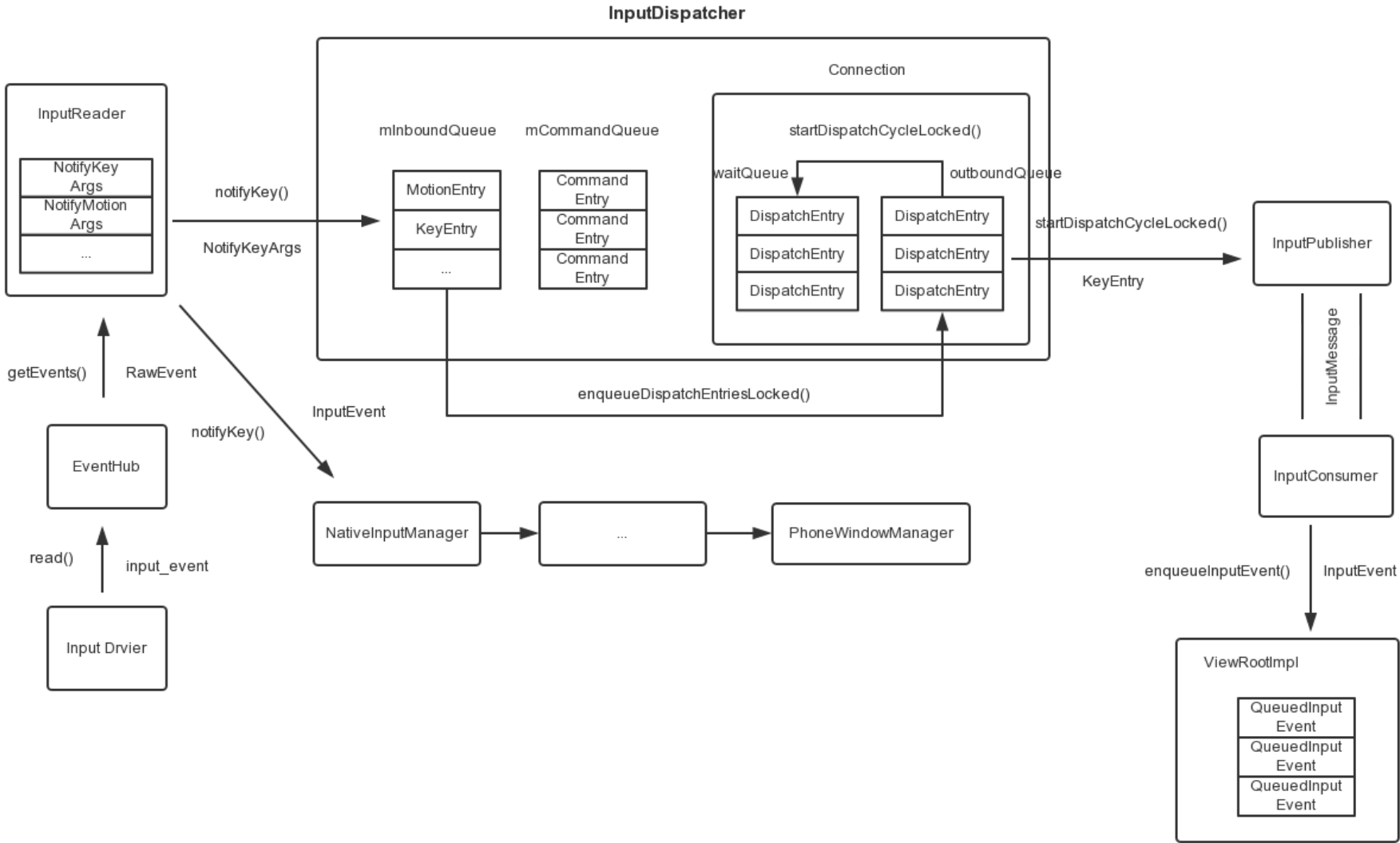
这里的mPolicy其实是NativeInputManager。NativeInputManager会通过JNI调用到Java世界中的IMS的相应函数：

```
3 | 1465         KeyEvent event, int policyFlags) {
                                     4 |
1466         return mWindowManagerCallbacks.interceptKeyBeforeDispatching(focus, event, policyFlags); 5 | 1467     }
```

这里的mWindowManagerCallbacks其实是InputMonitor。然后就调用到PWM了。

```
1 | 380     public long interceptKeyBeforeDispatching(
2 | 381         InputWindowHandle focus, KeyEvent event, int policyFlags) {
3 | 382         WindowState windowState = focus != null ? (WindowState) focus.windowState : null;
4 | 383         return mService.mPolicy.interceptKeyBeforeDispatching(windowState, event, policyFlags);
5 | 384     }
```

好了，到这里可以小结一下了。输入事件从Kernel到App和PWM的旅程大体如下图所示：

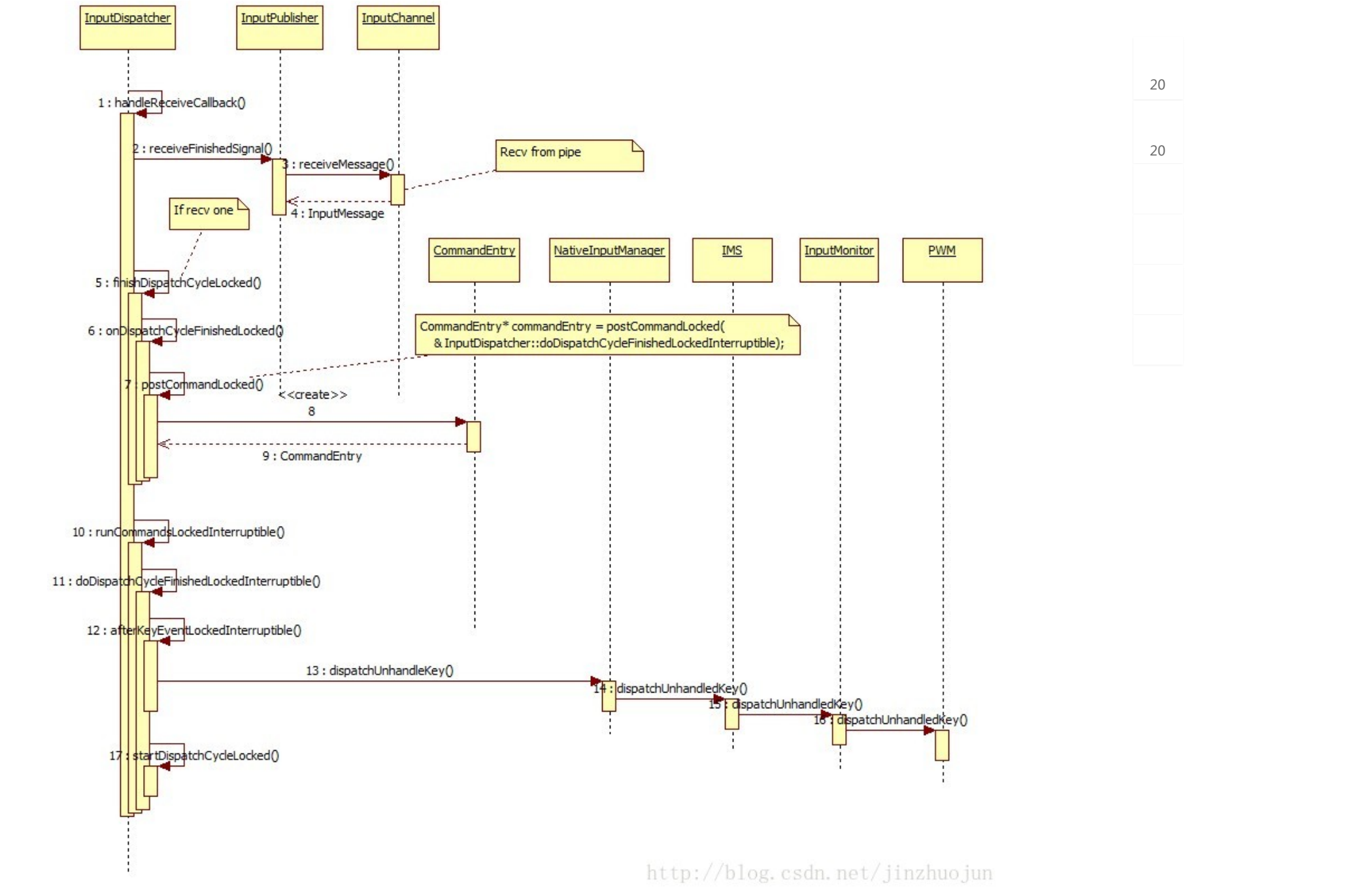


<http://blog.csdn.net/jinzhuojun>

可以看到，从Input driver中读出的是input\_event结构，之后在经过中间模块时会按需对其进行封装，变成或被封装进RawEvent, KeyEvent, DispatchEntry, InputMessage, InputEvent, QueuedInputEvent等结构。其中有些结构是对另一些结构的封装，如DispatchEntry中封装了KeyEvent，QueuedInputEvent是对InputEvent的封装等。

回到主线，故事来没讲完。上面说到App这端处理完输入事件，然后通过和IMS中InputDispatcher的通信管道InputChannel发了处理完成通知。那InputDispatcher这边收到后如何处理呢？





InputDispatcher会调用handleReceiveCallback()来处理完成信号。这里先是往Command队列里放一个处理事务执行doDispatchCycleFinishedLockedInterruptible()，后面在runCommandsLockedInterruptible()中会取出执行。在doDispatchCycleFinishedLockedInterruptible()函数中，会先调用afterKeyEventLockedInterruptible()。Android中可以定义一些Fallback键，即如果一个Key事件App没有处理，可以Fallback成另外默认的Key事件，这是在这里的dispatchUnhandledKey()函数中处理的。接着InputDispatcher会将该收到完成信号的事件项从等待队列中移除。同时由于上一个事件已被App处理完，就可以调用startDispatchCycleLocked()来进行下一轮事件的处理了。

```
1 3558 // Dequeue the event and start the next cycle.
2 3559 // Note that because the lock might have been released, it is possible that the
3 3560 // contents of the wait queue to have been drained, so we need to double-check
4 3561 // a few things.
5 3562 if (dispatchEntry == connection->findWaitQueueEntry(seq)) {
6 3563     connection->waitQueue.dequeue(dispatchEntry);
7 ...
8 3571 }
9 3572
10 3573 // Start the next dispatch cycle for this connection.
11 3574 startDispatchCycleLocked(now(), connection);
```

startDispatchCycleLocked函数会检查相应连接的输出缓冲中(connection->outboundQueue)是否有事件要发送的，有的话会通过InputChannel发送出去。

想对作者说点什么

baidu\_27798993： 很棒的文章 （1年前 #20楼）

👍0

中心世界： 赞 （1年前 #19楼）

👍0

Vincent\_Song： 我是小米的工程师，看到你的Android相关技术的博客写很好，现在小米Framework相关的工作机会，如果感兴趣请私信我你的联系方式，如有打扰，还请见谅，谢谢！ （1年前 #18楼）

👍0

登录

查看 20 条热评

expat介绍文档翻译

原文地址：http://www.xml.com/pub/a/1999/09/expat/index.html因为需要用，所以才翻译了这个文档。但总归...

博文 来自： ymj7150697的专栏

阅读数 3万+

jquery/js实现一个网页同时调用多个倒计时(最新的)

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,基本上都是千篇一律的不...

博文 来自： Websites

阅读数 44万+

关于SpringBoot bean无法注入的问题（与文件包位置有关）

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modulespring-boot-baseservice和d...

博文 来自： 开发随笔

阅读数 17万+

Android输入子系统浅析（一）

Linux输入子系统框架

博文 来自： \_我爱吃咸菜的专栏

阅读数 1697

深入理解Android输入系统

《深入理解Android卷III》即将发布，作者是张大伟。此书填补了深入理解AndroidFramework卷中的一个主要空白...

博文 来自： byron\_songxue

阅读数 3896

Linux/Android——输入子系统input\_event传递(二)

在前文Linux/Android——usb触摸屏驱动-usbtouchscreen中记录了如何在kernel中添加inputdevice类型为touch...

博文 来自： jscese

阅读数 1万+

Android输入子系统之InputReader读取键盘消息过程分析

InputReader读取键盘消息过程分析在Android输入子系统之启动过程分析中，InputManagerService启动之后，会...

博文 来自： chenweiaiyanyan...

阅读数 853

Android处理输入事件的流程（一）

我一直觉得要想学习Android，我们有必要研究一下Android的输入子系统，Android手机最主要的输入是触摸屏和...

博文 来自： cyz\_1257的博客

阅读数 1617

Android 事件输入系统整体框架 - 学无止境,静下心来! - CSDN博客

11-12

native层上报的事件消息和刷屏会先到达ViewRoot。对于刷屏事件,View...Android 5.0(Lollipop)事件输入系统(Input System) - 世事难料,保持低调 12-1...

Android Input System分析(一)--基本架构 - u013543848...\_CSDN博客

10-21

Android 5.0(Lollipop)事件输入系统(Input System) 12-13 1.2万 其实Android...来自: 世事难料,保持低调 Android底层开发入门(8)-InputSystem 10-15 1088 ...



Tamic大白

118篇文章

排名:5000+

关注



小码哥\_WS

135篇文章

排名:千里之外

关注



阳光玻璃杯

112篇文章

排名:8000+

关注



专注自我所爱

7篇文章

排名:千里之外

关注

Android5.0风格EditText输入框效果

一、android5.0上面有一个EditText填充效果：点击EditText之后，其中hint文字会向上面走动。然后可以填写内容...

博文 来自： u013470176的专栏

阅读数 3352

android 输入事件 - qq\_26825819的博客 - CSDN博客

11-24

一.提出问题 android是基于linux kernel的,linux的事件获取需要读/dev/input下的... Android 5.0(Lollipop)事件输入系统(Input System) - 世事难料,保持低调 12...

Android 输入管理服务-输入事件到达之后的处理流程 - m...\_CSDN博客

11-13

Android 输入事件系统之 EventHub 和 Input Lib(事件解析库)

从 Android事件输入系统整体框架 一文可知InputLibs是一个事件解析库，完成事件解析、keycode转换，设备配置...

博文 来自： 学无止境，静下心...

阅读数 1602

Android 5.0 Input初始化

android\frameworks\base\services\java\com\android\server\SystemServer.javaSlog.i(TAG,"InputManager")...

博文 来自： lyq284884的博客

阅读数 238

Android 输入事件系统之 EventHub 和 Input Lib(事件解...\_CSDN博客

12-6

Android 5.0(Lollipop)事件输入系统(Input System) 《-- 推荐阅读这篇 10-19...来自: 世事难料,保持低调 EventHub 09-19 446 在EventHub的构造函数中: ...

Android 5.0新控件——TextInputLayout - kongqw - CSDN博客

4-5

Android输入事件从读取到分发一：是谁在读取输入事件

零.第一次尝试阅读android输入系统的代码，免不了理解错误，如有错误，欢迎指正。一.提出问题android是基于lin...

博文 来自： 阳光玻璃杯

阅读数 2426

Android 从输入设备获取消息

Android消息获取过程概述Android输入系统架构

博文 来自： leif 的博客

阅读数 1407

Android输入管理InputManager之InputDispatcher得到事...\_CSDN博客

3-6

android 事件派发流程详解 05-27 阅读数 678 Android5.0(Lollipop)事件输入系统(InputSystem)2014-12-15 23 个评论 来源:世事难料,保持...

android输入子系统(以矩阵按键为例) - wangtt - CSDN博客

3-19

登录

注册

×

Android Input System分析（三）--Native

本来想跟大家讲一下设备节点的，后来发现这方面的资料很多，大家可以到网站自行搜索一下就可以了。在linux系统...

博文 来自： u013543848的博客

Android中InputManagerService里的InputReader和inputDispatcher

最近工作中遇到InputManagerService相关的问题，所以将InputReader和InputDispatcher看了一遍，水平有限，...

博文 来自： 浩海天空

mtk输入子系统键盘事件处理流程

MT6572平台来看一、输入子系统得到事件信息输入子系统首先由systemserver启动：inputManager=newInputM...

博文 来自： ollins136的专栏

Android5.0源码（Lollipop）

安卓5.0源码，个人路径下有安卓各版本的源码下载链接。

源码角度分析Android的事件输入系统（input system）及ANR原理分析

此篇我们从android5.0（lolipop）源码角度分析Android的事件输入系统（inputsystem）：先引用一张图来说明下...

博文 来自： hx\_401的专栏

Android Lollipop 5.0 经典新特性回顾

\*Tamic专注移动开发！更多文章请关注http://blog.csdn.net/sk719887916虽然Android已到了7.0，但是我们还是...

博文 来自： Tamic（大白）

5种优化你的安卓5.0 Lollipop代码的方法

简介原文地址Android5.0Lollipop\*发布的同时，推出了名为ART\*（简称Android运行时）的创新型默认运行时环境...

博文 来自： Shi\_Liz的博客

Android 4.0 事件输入(Event Input)系统

Android 4.0 事件输入(Event Input)系统

非局部均值去噪（NL-means）

非局部均值（NL-means）是近年来提出的一项新型的去噪技术。该方法充分利用了图像中的冗余信息，在去噪的同...

博文 来自： xiaoluo91的专栏

【深入Java虚拟机】之五：多态性实现机制——静态分派与动态分派

Class文件的编译过程中不包含传统编译中的连接步骤，一切方法调用在Class文件里面存储的都只是符号引用，而不...

博文 来自： 兰亭风雨的专栏

android底层驱动学习之linux输入子系统的理解

1.什么叫输入子系统？ 内核的输入子系统是对分散的，多种不同类别的输入设备(如键盘，鼠标，跟踪球，操纵杆，...

博文 来自： happyguys12345...

Android 输入子系统

input服务的启动流程1.android启动的时候会启动很多个service，参考SystemServer.java,会启动InputManagerSe...

博文 来自： gbmaotai的博客

Android输入子系统概览

Android输入子系统概览平台：Android6.0由于工作是基于Android系统做智能电视，因此平常的工作中在输入系统...

博文 来自： chenweiaiyanyan...

Android InputFlinger简单分析(主要分析Touch)

AndroidInputFlinger简单分析(主要分析Touch)首先，它有个服务，InputManagerService.InputManagerService...

博文 来自： bberdong的专栏

Android4.1 Framwork层Input子系统分发InputEvent流程图

今天整理了一下Android4.1的代码流程，将Input事件分发这部分画成了流程图，流程图里面清晰的描述了Input事件...

博文 来自： 撒哈拉的绿洲

Android输入事件从读取到分发三：InputDispatcherThread线程分发事件的过程

分析完事件的读取后，东忙西忙，不知不觉已过去了快五个月了...也不是说没有事件把这部分源码分析完，而是实在...

博文 来自： 阳光玻璃杯

Android EditText文本框重新获得焦点和输入完成后的检查事件触发

需要在文本输入完成后触发事件，如下：reg\_username=(EditText)findViewById(R.id.reg\_username);reg\_userna...

博文 来自： windvally的专栏

Android输入系统概述

Android输入系统概述Android输入系统概述简介简介在Android系统中，输入事件是由WindowManagerService服...

博文 来自： chewbee的专栏

Android输入系统框架

0000000000000000000

博文 来自： haoyun1990的博客

Android输入系统简介

1、当输入设备接入到android设备并且可用的时候，Linux内核会在 / dev/input/下创建名为event0-n或者其他名...

博文 来自： 欢迎关注微信公众...

Android 5.0.1（Lollipop）源码的下载、编译（eng版本）并烧录

转载：Android5.0.1（Lollipop）源码的下载、编译（eng版本）并烧录到Ne，有需要的朋友可以参考下。1.配置Li...

博文 来自： small5e4444的专栏

Android 5.0 Lollipop中新的Activity过渡效果

Android 5.0 Lollipop中新的Activity过渡效果,



Android5.0系统的优缺点

AndroidL ( 5.0 ) 正式定名为Lollipop(棒棒糖).安卓已经六岁了，也总算有一次重大改观了。安卓5.0Lollipop带来了...

博文 来自： [LYJ的IT生活](#)

阅读数 3985

练习向：Xposed安装和使用的踩坑

1.Xposed简介Xposed是一款优秀的androidjava层hook框架。它允许你在不修改apk源码的情况下，通过编写自己...

博文 来自： [ONS\\_cukuyo的博客](#)

阅读数 394

Android input子系统整体框架

2.模块结构下图是input输入子系统框架，输入子系统linux层由输入子系统核心层（ Core层 ），驱动层和事件处理...

博文 来自： [cjwsimple](#)

阅读数 357

Input子系统架构包括内核层与框架层详解

android input inputFlinger 第1章 Android Input子系统架构

12-01

下载

Android Input命令

AndroidInput命令inputinput是Android系统中的一个特殊的命令，用于模拟遥控器、键盘、鼠标的各种按键操作。...

博文 来自： [SuperLi](#)

阅读数 1135

Android Input流程分析（三）：InputReader

回到InputReader的loopOnce函数。 现在getEvents捞上来的RawEvent均保存在mEventBuffer中。/native/se...

博文 来自： [Invoker123的博客](#)

阅读数 764

Android 开发环境配置--实现第一个app “hello world！”

本文章写给想要学Android开发，但是苦恼于环境配置的小白，主要内容为android开发环境配置，实现自己的第一...

博文 来自： [StrongbyTime的...](#)

阅读数 1万+

Android 5.0(Lollipop)事件输入系统(Input System) 《-- 推荐阅读这篇

http://blog.csdn.net/jinzhuojun/article/details/41909159其实Android5.0中事件输入子系统的框架和流程没有本...

博文 来自： [thinkinwm的专栏](#)

阅读数 2012

Android之Input子系统事件分发流程

一、Android4.2系统服务侧1.服务端注册过程frameworks/base/core/java/android/view/ViewRootImpl.javapubl...

博文 来自： [程序改变生活](#)

阅读数 2万+

Android4.0 input事件输入流程详解(中间层到应用层)

在Android系统中，类似于键盘按键、触摸屏等事件是由WindowManagerService服务来管理的，然后再以消息的...

博文 来自： [撒哈拉的绿洲](#)

阅读数 6900

Android之input系统流程

按键或者触摸屏输入设备是最常用不过的设备，那么如果一个按键信息是如何从内核传递到android的呢，首先我们...

博文 来自： [leerobin83的专栏](#)

阅读数 1万+

基于PyTorch的深度学习入门教程（六）——数据并行化

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](#)

阅读数 4720

Android实现QQ分享及注意事项

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](#)

阅读数 5547

CNN笔记：通俗理解卷积神经网络

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](#)

阅读数 21万+

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

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...](#)

阅读数 9291

【小程序】微信小程序开发实践

帐号相关流程注册范围 企业 政府 媒体 其他组织换句话说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](#)

阅读数 25万+

HttpClient使用详解

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](#)

阅读数 80万+

android客户端与服务器端交互 如何保持session

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](#)

阅读数 4万+

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](#)

阅读数 1万+

如何在ArcGIS Online中构建自己的应用程序模板初级篇-显示地图

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...](#)

阅读数 4万+

再谈iOS 7的手势滑动返回功能

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](#)

阅读数 8万+

[ASP.NET]二维码的创建

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](#)

阅读数 5439



<div>webService学习（二）—— 调用自定义对象参数</div> <div>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...</div>	<div>博文</div> <div>来自：<a href="#">止水的专栏</a></div>	<div>阅读数 2万+</div>	
		20	
<div>人脸检测工具face_recognition的安装与应用</div> <div>人脸检测工具face_recognition的安装与应用</div>	<div>博文</div> <div>来自：<a href="#">roguesir的博客</a></div>	<div>阅读数 5万+</div>	20
<div>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）</div> <div>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...</div>	<div>博文</div> <div>来自：<a href="#">kongwei521的专栏</a></div>	<div>阅读数 2万+</div>	
<div>eclipse复制粘贴卡死</div> <div>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</div>	<div>博文</div> <div>来自：<a href="#">寒尘的专栏</a></div>	<div>阅读数 2488</div>	
<div>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</div> <div>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...</div>	<div>博文</div> <div>来自：<a href="#">Marswill</a></div>	<div>阅读数 14万+</div>	
<div>linux上安装Docker(非常简单的安装方法)</div> <div>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：...</div>	<div>博文</div> <div>来自：<a href="#">我走小路的博客</a></div>	<div>阅读数 20万+</div>	
<div>如何在ubuntu 16.04上安装 RealSense（相机型号：Intel SR300）</div> <div>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...</div>	<div>博文</div> <div>来自：<a href="#">z17816876284的...</a></div>	<div>阅读数 3633</div>	
<div>openfire 3.8.2 源码部署 /开发配置 / 二次开发</div> <div>最近新搞了openfire 从网上找了很多源码部署的相关文章但都是大同小异，拷贝加修改，我如是按照各个文章版本部...</div>	<div>博文</div> <div>来自：<a href="#">StillCity的专栏</a></div>	<div>阅读数 6643</div>	
<div>强连通分量及缩点tarjan算法解析</div> <div>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...</div>	<div>博文</div> <div>来自：<a href="#">九野的博客</a></div>	<div>阅读数 57万+</div>	
<div>【HTTP】Fiddler（一） - Fiddler简介</div> <div>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...</div>	<div>博文</div> <div>来自：<a href="#">专注、专心</a></div>	<div>阅读数 30万+</div>	
<div>OpenCV+OpenGL 双目立体视觉三维重建</div> <div>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...</div>	<div>博文</div> <div>来自：<a href="#">shiter编写程序的艺...</a></div>	<div>阅读数 4万+</div>	
<div>mybatis一级缓存(session cache)引发的问题</div> <div>mybatis一级缓存(session cache)引发的问题</div>	<div>博文</div> <div>来自：<a href="#">flysharkym的专栏</a></div>	<div>阅读数 2万+</div>	
<div>python图片处理类之~PIL.Image模块(ios android icon图标自动生成处理)</div> <div>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...</div>	<div>博文</div> <div>来自：<a href="#">专注于cocos+unit...</a></div>	<div>阅读数 5万+</div>	
<div>Libusb库在Android下的使用例程</div> <div>转载请注明：http://blog.csdn.net/hubbybob1/article/details/54863662 阅读本文前请先了解相关基础内容，操...</div>	<div>博文</div> <div>来自：<a href="#">hubbybob1专栏</a></div>	<div>阅读数 6807</div>	
<div>Hadoop+HBase完全分布式安装</div> <div>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...</div>	<div>博文</div> <div>来自：<a href="#">Dobbin</a></div>	<div>阅读数 4313</div>	
<div>DataTables 的 实例 《一》</div> <div>1.加载需要的js/css文件 2. function del(id){ alert(id); } var table; \$(document).ready(function(...</div>	<div>博文</div> <div>来自：<a href="#">辛修灿的博客</a></div>	<div>阅读数 1万+</div>	
<div>SpringAOP拦截Controller,Service实现日志管理(自定义注解的方式)</div> <div>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...</div>	<div>博文</div> <div>来自：<a href="#">czmchen的专栏</a></div>	<div>阅读数 12万+</div>	
<div>Android平台Camera实时滤镜实现方法探讨(五)--GLSurfaceView实现Camera预览</div> <div>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...</div>	<div>博文</div>	<div>阅读数 2万+</div>	
<div>JavaWeb多文件上传及zip打包下载</div> <div>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...</div>	<div>博文</div> <div>来自：<a href="#">kidQ的博客</a></div>	<div>阅读数 6536</div>	
<div>R语言逻辑回归、ROC曲线和十折交叉验证</div> <div>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...</div>	<div>博文</div> <div>来自：<a href="#">Tiaaaaaa的博客</a></div>	<div>阅读数 5万+</div>	
<div>CAVLC系数矩阵解析    统计学稳健估计opencv函数    设计制作学习    机器学习教程    Objective-C培训</div>			
<div>ios获取idfa    android title搜索    server的安全控制模型是什么 sql    ios 动态修改约束    系统深入学习java    ios系统培训课件</div>			

博客专家

原创

206

粉丝

1048

喜欢

314

评论

534

等级：

博客 7

访问：

134万+

积分：

1万+

排名：

1338

最新文章

单机玩转神经网络架构搜索(NAS) - Auto-Keras学习笔记

自动驾驶平台Apollo 3.5阅读手记：Cyber RT中的协程（Coroutine）

从《西部世界》到GAIL（Generative Adversarial Imitation Learning）算法

神经网络架构搜索（Neural Architecture Search）杂谈

自动驾驶平台Apollo 3.0阅读手记：perception模块之lane post processing

个人分类

Academic

8篇

Algorithm

14篇

Develop

45篇

Linux

54篇

Memo

13篇

展开

EDU

学院

CSDN学院

JOB

企业招聘

CSDN企业招聘

QQ客服

客服论坛

kefu@csdn.net

400-660-0108

工作时间 8:30-22:00

关于我们

招聘

广告服务

网站地图

百度提供站内搜索 京ICP备19004658号

©1999-2019 北京创新乐知网络技术有限公司

网络110报警服务

经营性网站备案信息

北京互联网违法和不良信息举报中心

中国互联网举报中心

家长监护

20
20

登录注册×