# Android之Input子系统事件分发流程

原创 2013年11月29日 08:49:08 19478

3

Android创建窗口机制，请看如下转载：

http://blog.csdn.net/sfdev/article/details/9130527

一、Android4.2系统服务侧——与View关系

1.服务端channel注册过程

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

```cpp
[cpp]
public void setView(View view, WindowManager.LayoutParams attrs, View panelParentView) {
  mInputChannel = new InputChannel(); //创建InputChannel
  res = mWindowSession.addToDisplay(mWindow, mSeq, mWindowAttributes,
    getHostVisibility(), mDisplay.getDisplayId(),
    mAttachInfo.mContentInsets, mInputChannel);  //创建与上述InputChannel对应的通道至服务端
  /*
  mWindowSession = WindowManagerGlobal.getWindowSession(context.getMainLooper());
  frameworks/base/core/java/android/view/WindowManagerGlobal.java
  public static IWindowSession getWindowSession(Looper mainLooper) {
    IWindowManager windowManager = getWindowManagerService();
    sWindowSession = windowManager.openSession(
                            imm.getClient(), imm.getInputContext());
    return sWindowSession;
  }
  frameworks/base/services/java/com/android/server/wm/WindowManagerService.java
  public IWindowSession openSession(IInputMethodClient client,
            IInputContext inputContext) {
    if (client == null) throw new IllegalArgumentException("null client");
    if (inputContext == null) throw new IllegalArgumentException("null inputContext");
    Session session = new Session(this, client, inputContext);
    return session;
  }
  */
  mInputEventReceiver = new WindowInputEventReceiver(mInputChannel,
    Looper.myLooper());  //将本通道注册进InputEventReceiver
}
```

frameworks/base/services/java/com/android/server/wm/Session.java

```cpp
[cpp]
public int addToDisplay(IWindow window, int seq, WindowManager.LayoutParams attrs,
            int viewVisibility, int displayId, Rect outContentInsets,
            InputChannel outInputChannel) {
  return mService.addWindow(this, window, seq, attrs, viewVisibility, displayId,
                outContentInsets, outInputChannel);
}
```

frameworks/base/services/java/com/android/server/wm/WindowManagerService.java

```cpp
[cpp]
public int addWindow(Session session, IWindow client, int seq,
            WindowManager.LayoutParams attrs, int viewVisibility, int displayId,
            Rect outContentInsets, InputChannel outInputChannel) {
  //以下包括了管道的创建（用于WMS与应用程序View通信）等
  String name = win.makeInputChannelName();
  InputChannel[] inputChannels = InputChannel.openInputChannelPair(name);
  win.setInputChannel(inputChannels[0]);
  inputChannels[1].transferTo(outInputChannel);
  //以下便是注册至server端过程
  //final InputManagerService mInputManager;
  mInputManager.registerInputChannel(win.mInputChannel, win.mInputWindowHandle);
}
```

frameworks/base/service/java/com/android/server/input/InputManagerService.java

```cpp
[cpp]
public void registerInputChannel(InputChannel inputChannel,
            InputWindowHandle inputWindowHandle) {
  nativeRegisterInputChannel(mPtr, inputChannel, inputWindowHandle, false);
```

tankai19880619
关注

| 原创 | 粉丝 | 喜欢 |
|---|---|---|
| 291 | 391 | 6 |

等级： 博客 7 访问量： 9
积分： 1万+ 排名： 174

## 他的最新文章

Android音量调节
Android录音机应用
Android的Audio子系统
Linux C多线程编程注意事项及数
Linux下缓冲区溢出攻击的原理及

## 文章分类

C/C++基础
汇编学习
java基础
数据结构与算法
面向对象之设计模式
软件工程
展开

## 文章存档

2016年8月
2016年7月
2016年4月
2016年2月
2016年1月
2015年12月
展开

## 他的热门文章

```
}
private static native void nativeRegisterInputChannel(int ptr, InputChannel inputChannel,
            InputWindowHandle inputWindowHandle, boolean monitor);
```

frameworks/base/service/jni/com_android_server_input_InputManagerService.cpp

[cpp]

```cpp
static void nativeRegisterInputChannel(JNIEnv* env, jclass clazz,
        jint ptr, jobject inputChannelObj, jobject inputWindowHandleObj, jboolean monitor) {
  NativeInputManager* im = reinterpret_cast<NativeInputManager*>(ptr);
  status_t status = im->registerInputChannel(
            env, inputChannel, inputWindowHandle, monitor);
}
status_t NativeInputManager::registerInputChannel(JNIEnv* env,
        const sp<InputChannel>& inputChannel,
        const sp<InputWindowHandle>& inputWindowHandle, bool monitor) {
  return mInputManager->getDispatcher()->registerInputChannel(
            inputChannel, inputWindowHandle, monitor);
  //mInputManager = new InputManager(eventHub, this, this);
  /*
  frameworks/base/services/input/InputManager.cpp
  sp<InputDispatcherInterface> InputManager::getDispatcher() {
    return mDispatcher;
  }
  mDispatcher = new InputDispatcher(dispatcherPolicy);
  */
}
```

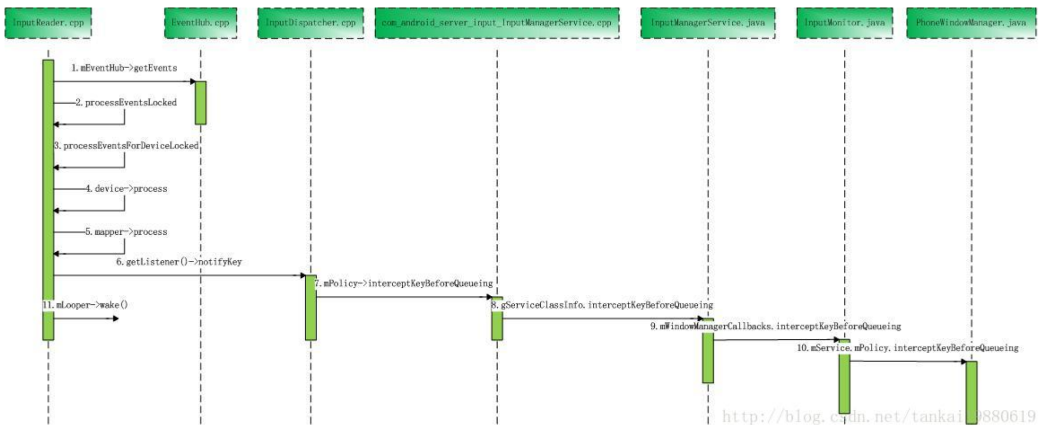
frameworks/base/services/input/InputDispatcher.cpp

[cpp]

```cpp
status_t InputDispatcher::registerInputChannel(const sp<InputChannel>& inputChannel,
        const sp<InputWindowHandle>& inputWindowHandle, bool monitor) {
  int fd = inputChannel->getFd();
  mConnectionsByFd.add(fd, connection);
  //该fd监听对应的处理函数为handleReceiveCallback
  mLooper->addFd(fd, 0, ALOOPER_EVENT_INPUT, handleReceiveCallback, this);
}
```

2.服务端上报过程

2.1.InputReaderThread线程从驱动读取数据并处理，如实现鼠标右键上报back键即在此处完成、以下代码将会看到


InputReader.cpp
EventHub.cpp
InputDispatcher.cpp
com_android_server_input_InputManagerService.cpp
InputManagerService.java
InputMonitor.java
PhoneWindowManager.java
1. mEventHub->getEvents
2. processEventsLocked
3. processEventsForDeviceLocked
4. device->process
5. mapper->process
6. getListener()->notifyKey
7. mPolicy->interceptKeyBeforeQueueing
8. gServiceClassInfo.interceptKeyBeforeQueueing
9. mWindowManagerCallbacks.interceptKeyBeforeQueueing
10. mService.mPolicy.interceptKeyBeforeQueueing
11. mLooper->wake()
http://blog.csdn.net/tankai19880619

frameworks/base/services/input/InputReader.cpp

[cpp]

```cpp
bool InputReaderThread::threadLoop() {
  mReader->loopOnce();
  return true;
}
void InputReader::loopOnce() {
  size_t count = mEventHub->getEvents(timeoutMillis, mEventBuffer, EVENT_BUFFER_SIZE);
  /*
  frameworks/base/services/input/EventHub.cpp
  size_t EventHub::getEvents(int timeoutMillis, RawEvent* buffer, size_t bufferSize) {
    int32_t readSize = read(device->fd, readBuffer,
      sizeof(struct input_event) * capacity);//从驱动读取事件
  }
  */
  processEventsLocked(mEventBuffer, count);
}
void InputReader::processEventsLocked(const RawEvent* rawEvents, size_t count) {
  processEventsForDeviceLocked(deviceId, rawEvent, batchSize);
```

3

Fatal signal 11问题的解决方法
84300

USB设备的VID与PID
35993

Android系统Camera录像过程分
25585

Android之Input子系统事件分发
19453

Android系统Camera参数设置
17451

《Linux启动过程分析》内核挂载
统
16489

《Linux启动过程分析》之区别Ir
与initrd
15774

Android WiFi之SoftAP
15685

Android5.0以太网流程源码情景
14689

Android5.0系统wifi—直在向wp
n问题
12990

联系我们

请扫描二维码联系
webmaster@
400-660-01
QQ客服

关于 招聘 广告服务

```
}
void InputReader::processEventsForDeviceLocked(int32_t deviceId,
        const RawEvent* rawEvents, size_t count) {
  device->process(rawEvents, count);
}
void InputDevice::process(const RawEvent* rawEvents, size_t count) {
  //该设备的所有mapper进行处理：注意：这里使用了多态
  for (size_t i = 0; i < numMappers; i++) {
    InputMapper* mapper = mMappers[i];
    mapper->process(rawEvent);
  }
}
//以下就是各个mapper
//CursorInput鼠标设备
void CursorInputMapper::process(const RawEvent* rawEvent) {
  mCursorButtonAccumulator.process(rawEvent);
  mCursorMotionAccumulator.process(rawEvent);
  mCursorScrollAccumulator.process(rawEvent);
  if (rawEvent->type == EV_SYN && rawEvent->code == SYN_REPORT) {
    sync(rawEvent->when);
  }
}
//CursorButtonAccumulator::process(const RawEvent* rawEvent)
//CursorMotionAccumulator::process(const RawEvent* rawEvent)
//CursorScrollAccumulator::process(const RawEvent* rawEvent)
void CursorInputMapper::sync(nsecs_t when) {
  int32_t currentButtonState = mCursorButtonAccumulator.getButtonState();
  /*
  uint32_t CursorButtonAccumulator::getButtonState() const {
    if (mBtnRight) {
      //Changed by tank for mouse left button to back
      result |= AMOTION_EVENT_BUTTON_BACK;
      //  result |= AMOTION_EVENT_BUTTON_SECONDARY;
    }
    if (mBtnMiddle) {
      //change by tank@tcl.com for mouse middle button to menu
      result |= AMOTION_EVENT_BUTTON_MENU;
      //result |= AMOTION_EVENT_BUTTON_TERTIARY;
    }
  }
  */

  getListener()->notifyMotion(&args);

  synthesizeButtonKeys(getContext(), AKEY_EVENT_ACTION_UP, when, getDeviceId(), mSource,
      policyFlags, lastButtonState, currentButtonState);
  /*
  static void synthesizeButtonKeys(InputReaderContext* context, int32_t action,
      nsecs_t when, int32_t deviceId, uint32_t source,
      uint32_t policyFlags, int32_t lastButtonState, int32_t currentButtonState) {
    synthesizeButtonKey(context, action, when, deviceId, source, policyFlags,
              lastButtonState, currentButtonState,
              AMOTION_EVENT_BUTTON_BACK, AKEYCODE_BACK);
    synthesizeButtonKey(context, action, when, deviceId, source, policyFlags,
              lastButtonState, currentButtonState,
              AMOTION_EVENT_BUTTON_FORWARD, AKEYCODE_FORWARD);
    //add by tank  mouse key event middle->menu.
    synthesizeButtonKey(context, action, when, deviceId, source, policyFlags,
              lastButtonState, currentButtonState,
              AMOTION_EVENT_BUTTON_MENU, AKEYCODE_MENU);
    //end tank
  }
  static void synthesizeButtonKey(InputReaderContext* context, int32_t action,
        nsecs_t when, int32_t deviceId, uint32_t source,
        uint32_t policyFlags, int32_t lastButtonState, int32_t currentButtonState,
        int32_t buttonState, int32_t keyCode) {
    if ((action == AKEY_EVENT_ACTION_DOWN && !(lastButtonState & buttonState)
    && (currentButtonState & buttonState))
    || (action == AKEY_EVENT_ACTION_UP
    && (lastButtonState & buttonState)
    && !(currentButtonState & buttonState))) {
      context->getListener()->notifyKey(&args);
    }
  }
  */
}
//TouchInput触摸板设备
void SingleTouchInputMapper::process(const RawEvent* rawEvent)
  TouchInputMapper::process(rawEvent);
  mSingleTouchMotionAccumulator.process(rawEvent);
}
//SingleTouchMotionAccumulator::process(const RawEvent* rawEvent)
void MultiTouchInputMapper::process(const RawEvent* rawEvent) {
  TouchInputMapper::process(rawEvent);
  mMultiTouchMotionAccumulator.process(rawEvent);
}
//MultiTouchMotionAccumulator::process(const RawEvent* rawEvent)
```

```cpp
void TouchInputMapper::process(const RawEvent* rawEvent) {
  mCursorButtonAccumulator.process(rawEvent);
  mCursorScrollAccumulator.process(rawEvent);
  mTouchButtonAccumulator.process(rawEvent);
  if (rawEvent->type == EV_SYN && rawEvent->code == SYN_REPORT) {
    sync(rawEvent->when);
  }
}
//TouchButtonAccumulator::process(const RawEvent* rawEvent)
void TouchInputMapper::sync(nsecs_t when) {
  dispatchTouches(when, policyFlags);
}
void TouchInputMapper::dispatchTouches(nsecs_t when, uint32_t policyFlags) {
  dispatchMotion(when, policyFlags, mSource,
    AMOTION_EVENT_ACTION_MOVE, 0, metaState, buttonState,
    AMOTION_EVENT_EDGE_FLAG_NONE,
    mCurrentCookedPointerData.pointerProperties,
    mCurrentCookedPointerData.pointerCoords,
    mCurrentCookedPointerData.idToIndex,
    currentIdBits, -1,
    mOrientedXPrecision, mOrientedYPrecision, mDownTime);
}
void TouchInputMapper::dispatchMotion(nsecs_t when, uint32_t policyFlags, uint32_t source,
    int32_t action, int32_t flags, int32_t metaState, int32_t buttonState, int32_t edgeFlags,
    const PointerProperties* properties, const PointerCoords* coords,
    const uint32_t* idToIndex, BitSet32 idBits,
    int32_t changedId, float xPrecision, float yPrecision, nsecs_t downTime) {
  getListener()->notifyMotion(&args);
}
//SwitchInput设备
void SwitchInputMapper::process(const RawEvent* rawEvent) {
  sync(rawEvent->when);
}
void SwitchInputMapper::sync(nsecs_t when) {
  getListener()->notifySwitch(&args);
}
//JoystickInput游戏手柄设备
void JoystickInputMapper::process(const RawEvent* rawEvent) {
  sync(rawEvent->when, false /*force*/);
}
void JoystickInputMapper::sync(nsecs_t when, bool force) {
  getListener()->notifyMotion(&args);
}
//KeyboardInput按键设备
void KeyboardInputMapper::process(const RawEvent* rawEvent) {
  processKey(rawEvent->when, rawEvent->value != 0, keyCode, scanCode, flags);
}
void KeyboardInputMapper::processKey(nsecs_t when, bool down, int32_t keyCode,
        int32_t scanCode, uint32_t policyFlags) {
  getListener()->notifyKey(&args);
}
```

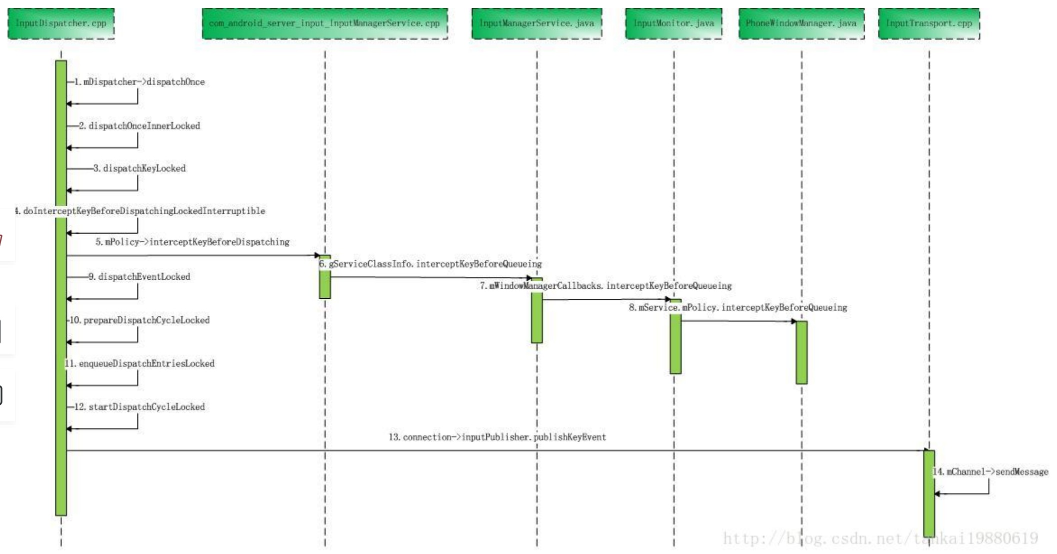
2.2.InputReaderThread线程对系统层按键做处理（比较重要的是POWER键，最终在PhoneWindowManager中的interceptKeyBeforeQueueing和interceptMotionBeforeQueueingWhenScreenOff）后分发给InputDispatcherThread线程，以下分析将看到之前一个鼠标操作过程中无法待机的问题解决

以下几种情况都会唤醒InputDispatcherThread线程，即调用mLooper->wake()唤醒正在awoken()中的InputReaderThread线程：

frameworks/base/services/input/InputDispatcher.cpp

```cpp
[cpp]
//有新输入设备注册等
void InputDispatcher::notifyConfigurationChanged(const NotifyConfigurationChangedArgs* args) {
  ConfigurationChangedEntry* newEntry = new ConfigurationChangedEntry(args->eventTime);
  needWake = enqueueInboundEventLocked(newEntry);
  if (needWake) {
    mLooper->wake();
  }
}
//分发按键事件
void InputDispatcher::notifyKey(const NotifyKeyArgs* args) {
  //说明：PhoneWindowManager.java中policyFlags位决定系统按键（如HOME等是否需要由系统处理）
  mPolicy->interceptKeyBeforeQueueing(&event, policyFlags);
  //以下分析将看到，该调用实际是在PhoneWindowManager.java中实现
  /*
  frameworks/base/services/input/InputManager.cpp
  InputManager::InputManager(
        const sp<EventHubInterface>& eventHub,
        const sp<InputReaderPolicyInterface>& readerPolicy,
        const sp<InputDispatcherPolicyInterface>& dispatcherPolicy) {
    mDispatcher = new InputDispatcher(dispatcherPolicy);
    mReader = new InputReader(eventHub, readerPolicy, mDispatcher);
  }
  frameworks/base/services/jni/com_android_server_input_InputManagerService.cpp
  NativeInputManager::NativeInputManager(jobject contextObj,
        jobject serviceObj, const sp<Looper>& looper) :
        mLooper(looper) {
    mInputManager = new InputManager(eventHub, this, this);
  }
  void NativeInputManager::interceptKeyBeforeQueueing(const KeyEvent* keyEvent,
        uint32_t& policyFlags) {
    wmActions = env->CallIntMethod(mServiceObj,
                    gServiceClassInfo.interceptKeyBeforeQueueing,
                    keyEventObj, policyFlags, isScreenOn);
    //如下函数中将有待机和开机的处理
    handleInterceptActions(wmActions, when, policyFlags);
  }
  frameworks/base/service/java/com/android/server/input/InputManagerService.java
  private int interceptKeyBeforeQueueing(KeyEvent event, int policyFlags, boolean isScreenOn) {
    return mWindowManagerCallbacks.interceptKeyBeforeQueueing(
              event, policyFlags, isScreenOn);
  }
  frameworks/base/service/java/com/android/server/SystemServer.java
  inputManager = new InputManagerService(context, wmHandler);
  wm = WindowManagerService.main(context, power, display, inputManager,
      uiHandler, wmHandler,
      factoryTest != SystemServer.FACTORY_TEST_LOW_LEVEL,
      !firstBoot, onlyCore);
  inputManager.setWindowManagerCallbacks(wm.getInputMonitor());
  frameworks/base/service/java/com/android/server/wm/WindowManagerService.java
  public InputMonitor getInputMonitor() {
    return mInputMonitor;
  }
  frameworks/base/service/java/com/android/server/wm/InputMonitor.java
  public int interceptKeyBeforeQueueing(
          KeyEvent event, int policyFlags, boolean isScreenOn) {
    return mService.mPolicy.interceptKeyBeforeQueueing(event, policyFlags, isScreenOn);
  }
  public InputMonitor(WindowManagerService service) {
    mService = service;
  }
  frameworks/base/service/java/com/android/server/wm/WindowManagerService.java
  final WindowManagerPolicy mPolicy = PolicyManager.makeNewWindowManager();
  frameworks/base/core/java/com/android/internal/policy/PolicyManager.java
  public static WindowManagerPolicy makeNewWindowManager() {
    return sPolicy.makeNewWindowManager();
  }
  private static final String POLICY_IMPL_CLASS_NAME =
        "com.android.internal.policy.impl.Policy";
  Class policyClass = Class.forName(POLICY_IMPL_CLASS_NAME);
  sPolicy = (IPolicy)policyClass.newInstance();
  frameworks/base/core/java/com/android/internal/policy/Policy.java
  package com.android.internal.policy.impl;
  public class Policy implements IPolicy {
    public WindowManagerPolicy makeNewWindowManager() {
      return new PhoneWindowManager();
    }
  }
  frameworks/base/core/java/com/android/internal/policy/PhoneWindowManager.java
  public int interceptKeyBeforeQueueing(KeyEvent event, int policyFlags, boolean isScreenOn) {
    case KeyEvent.KEYCODE_POWER: {
      result = (result & ~ACTION_WAKE_UP) | ACTION_GO_TO_SLEEP;
    }
  }
  */
  KeyEntry* newEntry = new KeyEntry(args->eventTime,
              args->deviceId, args->source, policyFlags,
              args->action, flags, args->keyCode, args->scanCode,
              metaState, repeatCount, args->downTime);
  needWake = enqueueInboundEventLocked(newEntry);
  if (needWake) {
    mLooper->wake();
  }
}
//分发Motion事件
void InputDispatcher::notifyMotion(const NotifyMotionArgs* args) {
  mPolicy->interceptMotionBeforeQueueing(args->eventTime, /*byref*/ policyFlags);
  /*
  如上分析，不再累赘；该接口是：
  frameworks/base/services/jni/com_android_server_input_InputManagerService.cpp
  void NativeInputManager::interceptMotionBeforeQueueing(nsecs_t when, uint32_t& policyFlags) {
    jint wmActions = env->CallIntMethod(mServiceObj,
                        gServiceClassInfo.interceptMotionBeforeQueueingWhenScreenOff,
                        policyFlags);
    handleInterceptActions(wmActions, when,  policyFlags);
  }
  如上interceptMotionBeforeQueueingWhenScreenOff在PhoneWindowManager中实现；分析同上，不再累赘：
  frameworks/base/core/java/com/android/internal/policy/PhoneWindowManager.java
  public int interceptMotionBeforeQueueingWhenScreenOff(int policyFlags) {
      //result |= ACTION_WAKE_UP;
      //add by tank
      result = result & (~ACTION_WAKE_UP);
      //end tank
      return result;
    }
    看看handleInterceptActions函数:
    void NativeInputManager::handleInterceptActions(jint wmActions, nsecs_t when,
          uint32_t& policyFlags) {
      //接上边PhoneWindowManager中interceptKeyBeforeQueueing对于power键的返回值可知，系统将待机
      if (wmActions & WM_ACTION_GO_TO_SLEEP) {
        #if DEBUG_INPUT_DISPATCHER_POLICY
        ALOGD("handleInterceptActions: Going to sleep.");
        #endif
        android_server_PowerManagerService_goToSleep(when);
      }
      //以下说明PhoneWindowManager中interceptMotionBeforeQueueingWhenScreenOff返回值WM_ACTION_WAKE_UP将会导致唤
醒
      //当然，是可是收到motion事件的前提下
      if (wmActions & WM_ACTION_WAKE_UP) {
        #if DEBUG_INPUT_DISPATCHER_POLICY
        ALOGD("handleInterceptActions: Waking up.");
        #endif
        android_server_PowerManagerService_wakeUp(when);
      }
      //以下是可以上报给系统的
      if (wmActions & WM_ACTION_PASS_TO_USER) {
          policyFlags |= POLICY_FLAG_PASS_TO_USER;
      }
    }
    */
    MotionEntry* newEntry = new MotionEntry(args->eventTime,
                  args->deviceId, args->source, policyFlags,
                  args->action, args->flags, args->metaState, args->buttonState,
                  args->edgeFlags, args->xPrecision, args->yPrecision, args->downTime,
                  args->displayId,
                  args->pointerCount, args->pointerProperties, args->pointerCoords);
    needWake = enqueueInboundEventLocked(newEntry);
    if (needWake) {
      mLooper->wake();
    }
  }
  //设备重置
  void InputDispatcher::notifyDeviceReset(const NotifyDeviceResetArgs* args) {
    DeviceResetEntry* newEntry = new DeviceResetEntry(args->eventTime, args->deviceId);
    needWake = enqueueInboundEventLocked(newEntry);
    if (needWake) {
      mLooper->wake();
    }
  }
  //C层的按键注入接口
  int32_t InputDispatcher::injectInputEvent(const InputEvent* event,
          int32_t injectorPid, int32_t injectorUid, int32_t syncMode, int32_t timeoutMillis,
          uint32_t policyFlags) {
    needWake |= enqueueInboundEventLocked(entry);
    if (needWake) {
      mLooper->wake();
    }
  }
  //setInputWindows
  //setFocusedApplication
  //setInputDispatchMode
  //setInputFilterEnabled
  //transferTouchFocus
  //registerInputChannel
  //unregisterInputChannel
  //monitor
```

2.3.InputDispatcherThread线程处理，根据PhoneWindowManager中的interceptKeyBeforeDispatching决定是否丢弃按键


InputDispatcher.cpp
com_android_server_input_InputManagerService.cpp
InputManagerService.java
InputMonitor.java
PhoneWindowManager.java
InputTransport.cpp
1. mDispatcher->dispatchOnce
2. dispatchOnceInnerLocked
3. dispatchKeyLocked
4. doInterceptKeyBeforeDispatchingLockedInterruptible
5. mPolicy->interceptKeyBeforeDispatching
6. gServiceClassInfo.interceptKeyBeforeQueueing
7. mWindowManagerCallbacks.interceptKeyBeforeQueueing
8. mService.mPolicy.interceptKeyBeforeQueueing
9. dispatchEventLocked
10. prepareDispatchCycleLocked
11. enqueueDispatchEntriesLocked
12. startDispatchCycleLocked
13. connection->inputPublisher.publishKeyEvent
14. mChannel->sendMessage
http://blog.csdn.net/tankai19880619

## InputDispatcherThread线程被唤醒

**[cpp]**

```cpp
bool InputDispatcherThread::threadLoop() {
  mDispatcher->dispatchOnce();
  return true;
}
void InputDispatcher::dispatchOnce() {
  dispatchOnceInnerLocked(&nextWakeupTime);
  mLooper->pollOnce(timeoutMillis);
}
void InputDispatcher::dispatchOnceInnerLocked(nsecs_t* nextWakeupTime) {
  if (!mPolicy->isKeyRepeatEnabled()) {
        resetKeyRepeatLocked();
  }
  switch (mPendingEvent->type) {
    case EventEntry::TYPE_CONFIGURATION_CHANGED: {
      done = dispatchConfigurationChangedLocked(currentTime, typedEntry);
    }
    case EventEntry::TYPE_DEVICE_RESET: {
      done = dispatchDeviceResetLocked(currentTime, typedEntry);
    }
    case EventEntry::TYPE_KEY: {
      done = dispatchKeyLocked(currentTime, typedEntry, &dropReason, nextWakeupTime);
    }
    case EventEntry::TYPE_MOTION: {
      done = dispatchMotionLocked(currentTime, typedEntry,
                  &dropReason, nextWakeupTime);
    }
  }
  dropInboundEventLocked(mPendingEvent, dropReason);  //丢弃的事件！！！！
}

bool InputDispatcher::dispatchKeyLocked(nsecs_t currentTime, KeyEntry* entry,
        DropReason* dropReason, nsecs_t* nextWakeupTime) {
  CommandEntry* commandEntry = postCommandLocked(
                  & InputDispatcher::doInterceptKeyBeforeDispatchingLockedInterruptible);
  /*
  void InputDispatcher::doInterceptKeyBeforeDispatchingLockedInterruptible(
        CommandEntry* commandEntry) {
    //说明：PhoneWindowManager.java中可以截断事件而不上报，即返回-1、将被丢弃
    nsecs_t delay = mPolicy->interceptKeyBeforeDispatching(commandEntry->inputWindowHandle,
            &event, entry->policyFlags);
    if (delay < 0) {
        entry->interceptKeyResult = KeyEntry::INTERCEPT_KEY_RESULT_SKIP;
    } else if (!delay) {
        entry->interceptKeyResult = KeyEntry::INTERCEPT_KEY_RESULT_CONTINUE;
    } else {
        entry->interceptKeyResult = KeyEntry::INTERCEPT_KEY_RESULT_TRY_AGAIN_LATER;
        entry->interceptKeyWakeupTime = now() + delay;
    }
  }
  */
  else if (entry->interceptKeyResult == KeyEntry::INTERCEPT_KEY_RESULT_SKIP) {
    if (*dropReason == DROP_REASON_NOT_DROPPED) {
      *dropReason = DROP_REASON_POLICY; //dropReason是因为策略丢弃
    }
  }
  if (*dropReason != DROP_REASON_NOT_DROPPED) {
    setInjectionResultLocked(entry, *dropReason == DROP_REASON_POLICY
      ? INPUT_EVENT_INJECTION_SUCCEEDED : INPUT_EVENT_INJECTION_FAILED);
    return true;
  }
  dispatchEventLocked(currentTime, entry, inputTargets);
}
bool InputDispatcher::dispatchMotionLocked(
        nsecs_t currentTime, MotionEntry* entry, DropReason* dropReason, nsecs_t* nextWakeupTime) {
  dispatchEventLocked(currentTime, entry, inputTargets);
}
```

3

## 2.4.InputDispatcherThread线程分发给应用程序进程

**在这里解决了up事件上报两次的问题！！！！！！**

rameworks/base/services/input/InputDispatcher.cpp

[cpp]

```cpp
void InputDispatcher::dispatchEventLocked(nsecs_t currentTime,
        EventEntry* eventEntry, const Vector<InputTarget>& inputTargets) {
  pokeUserActivityLocked(eventEntry);  //和Activity相关，后边三中有设备删除的分析：基本同下
  ssize_t connectionIndex = getConnectionIndexLocked(inputTarget.inputChannel);
  sp<Connection> connection = mConnectionsByFd.valueAt(connectionIndex);
  prepareDispatchCycleLocked(currentTime, connection, eventEntry, &inputTarget);
}
void InputDispatcher::prepareDispatchCycleLocked(nsecs_t currentTime,
        const sp<Connection>& connection, EventEntry* eventEntry, const InputTarget* inputTarget) {
  enqueueDispatchEntriesLocked(currentTime, connection, eventEntry, inputTarget);
}
void InputDispatcher::enqueueDispatchEntriesLocked(nsecs_t currentTime,
        const sp<Connection>& connection, EventEntry* eventEntry, const InputTarget* inputTarget) {
   enqueueDispatchEntryLocked(connection, eventEntry, inputTarget,
            InputTarget::FLAG_DISPATCH_AS_HOVER_EXIT); //将按键注入队列
   /*
   void InputDispatcher::enqueueDispatchEntryLocked(
        const sp<Connection>& connection, EventEntry* eventEntry, const InputTarget* inputTarget,
        int32_t dispatchMode) {
     DispatchEntry* dispatchEntry = new DispatchEntry(eventEntry, // increments ref
            inputTargetFlags, inputTarget->xOffset, inputTarget->yOffset,
            inputTarget->scaleFactor);
     if (!connection->inputState.trackKey(keyEntry,
                dispatchEntry->resolvedAction, dispatchEntry->resolvedFlags) || (dispatchEntry->resolvedFla
gs == 0x28)){
       //add by tankai 0x28
       delete dispatchEntry;
       return;
     }
   }
   */
   //dropInboundEventLocked
   //synthesizeCancelationEventsForAllConnectionsLocked->
   //synthesizeCancelationEventsForConnectionLocked->
   /*
   void InputDispatcher::synthesizeCancelationEventsForConnectionLocked(
        const sp<Connection>& connection, const CancelationOptions& options) {
     Vector<EventEntry*> cancelationEvents;
     connection->inputState.synthesizeCancelationEvents(currentTime,
            cancelationEvents, options);
     //关键在这里，mKeyMementos；在enqueueDispatchEntryLocked时调用trackKey由addKeyMemento注入！！！！！！
     if (!cancelationEvents.isEmpty()) {
       enqueueDispatchEntryLocked(connection, cancelationEventEntry, // increments ref
                 &target, InputTarget::FLAG_DISPATCH_AS_IS);
     }
   }
   */
   //enqueueDispatchEntriesLocked,注入了0x28标志的按键
   startDispatchCycleLocked(currentTime, connection);
}
void InputDispatcher::startDispatchCycleLocked(nsecs_t currentTime,
        const sp<Connection>& connection) {
  switch (eventEntry->type) {
    case EventEntry::TYPE_KEY: {
      status = connection->inputPublisher.publishKeyEvent(dispatchEntry->seq,
                    keyEntry->deviceId, keyEntry->source,
                    dispatchEntry->resolvedAction, dispatchEntry->resolvedFlags,
                    keyEntry->keyCode, keyEntry->scanCode,
                    keyEntry->metaState, keyEntry->repeatCount, keyEntry->downTime,
                    keyEntry->eventTime);
    }
    case EventEntry::TYPE_MOTION: {
      status = connection->inputPublisher.publishMotionEvent(dispatchEntry->seq,
                    motionEntry->deviceId, motionEntry->source,
                    dispatchEntry->resolvedAction, dispatchEntry->resolvedFlags,
                    motionEntry->edgeFlags, motionEntry->metaState, motionEntry->buttonState,
                    xOffset, yOffset,
                    motionEntry->xPrecision, motionEntry->yPrecision,
                    motionEntry->downTime, motionEntry->eventTime,
                    motionEntry->pointerCount, motionEntry->pointerProperties,
                    usingCoords);
    }
  }
}
```

frameworks/base/libs/androidfw/InputTransport.cpp

[cpp]

```cpp
status_t InputPublisher::publishKeyEvent(
        uint32_t seq,
        int32_t deviceId,
        int32_t source,
        int32_t action,
        int32_t flags,
        int32_t keyCode,
        int32_t scanCode,
        int32_t metaState,
        int32_t repeatCount,
        nsecs_t downTime,
        nsecs_t eventTime) {
  return mChannel->sendMessage(&msg);
}
status_t InputChannel::sendMessage(const InputMessage* msg) {
  do {
        nWrite = ::send(mFd, msg, msgLength, MSG_DONTWAIT | MSG_NOSIGNAL);
  } while (nWrite == -1 && errno == EINTR);
}
```

二、Android4.2系统应用程序侧——与View关系

InputManagerService也就是InputDispatcher与应用程序通信是靠looper。

说明：

  InputReader从设备文件中读取的是RawEvent，在交给InputDispatcher进行分发之前，它需要先把RawEvent进行转化分类，拆分成KeyEvent、MotionEvent、TrackEvent各种类型等。

  InputDispatcher获得按键事件后，根据当前设备的状况来优先消化事件（该过程交由PhoneWindowManager.java来处理）；最后，剩余事件分发给ViewRoot；ViewRoot再分发给IME输入法或View、Activity。

1.应用程序View中channel注册过程

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

[cpp]

```cpp
public void setView(View view, WindowManager.LayoutParams attrs, View panelParentView) {
  mInputChannel = new InputChannel(); //创建InputChannel
  res = mWindowSession.addToDisplay(mWindow, mSeq, mWindowAttributes,
    getHostVisibility(), mDisplay.getDisplayId(),
    mAttachInfo.mContentInsets, mInputChannel);  //创建与上述InputChannel对应的通道至服务端
  mInputEventReceiver = new WindowInputEventReceiver(mInputChannel,
    Looper.myLooper());  //将本通道注册进InputEventReceiver
}
final class WindowInputEventReceiver extends InputEventReceiver {
  public WindowInputEventReceiver(InputChannel inputChannel, Looper looper) {
    super(inputChannel, looper);
  }
  @Override
  public void onInputEvent(InputEvent event) {
    enqueueInputEvent(event, this, 0, true);
  }
}
```

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

[cpp]

```cpp
public InputEventReceiver(InputChannel inputChannel, Looper looper) {
  mReceiverPtr = nativeInit(this, inputChannel, mMessageQueue);
}
private static native int nativeInit(InputEventReceiver receiver,
            InputChannel inputChannel, MessageQueue messageQueue);
```

frameworks/base/core/jni/android_view_InputEventReceiver.cpp

[cpp]

```cpp
static jint nativeInit(JNIEnv* env, jclass clazz, jobject receiverObj,
        jobject inputChannelObj, jobject messageQueueObj) {
  sp<NativeInputEventReceiver> receiver = new NativeInputEventReceiver(env,
            receiverObj, inputChannel, messageQueue);
  status_t status = receiver->initialize();
}
status_t NativeInputEventReceiver::initialize() {
  int receiveFd = mInputConsumer.getChannel()->getFd();
  mMessageQueue->getLooper()->addFd(receiveFd, 0, ALOOPER_EVENT_INPUT, this, NULL);
  return OK;
}
```

rameworks/native/libs/utils/Looper.cpp

**[cpp]**

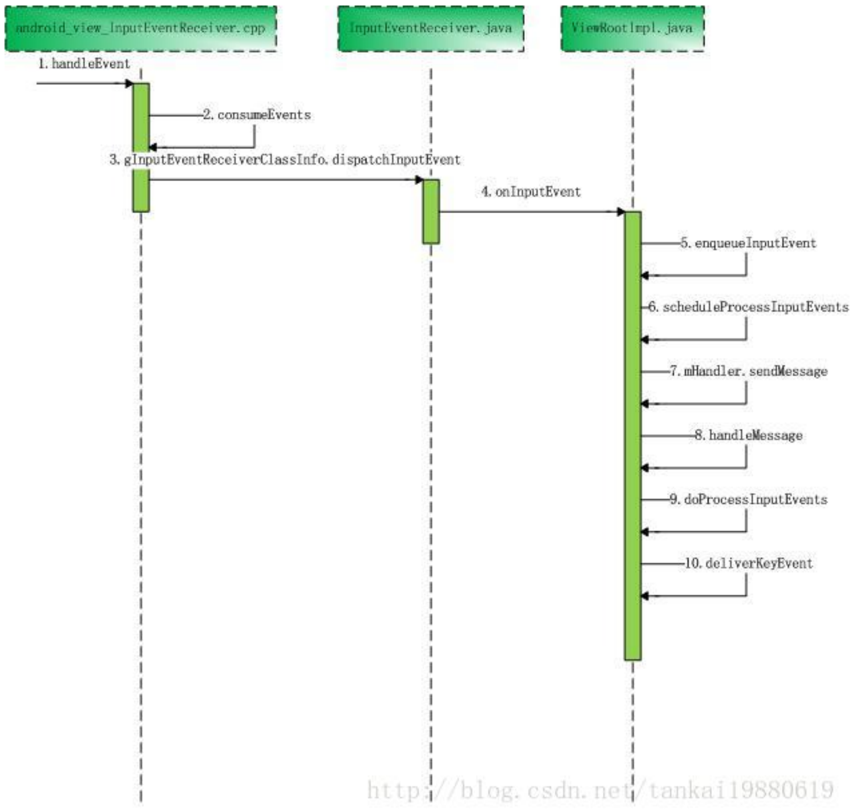
```cpp
int Looper::addFd(int fd, int ident, int events, const sp<LooperCallback>& callback, void* data) {
  request.callback = callback;
}
```

2.应用程序View响应过程

frameworks/native/libs/utils/Looper.cpp

**[cpp]**

```cpp
int Looper::pollInner(int timeoutMillis) {
  awoken(); //阻塞，等待
  int callbackResult = response.request.callback->handleEvent(fd, events, data);
}
```



frameworks/base/core/jni/android_view_InputEventReceiver.cpp

**[cpp]**

```cpp
int NativeInputEventReceiver::handleEvent(int receiveFd, int events, void* data) {
  status_t status = consumeEvents(env, false /*consumeBatches*/, -1);
}
status_t NativeInputEventReceiver::consumeEvents(JNIEnv* env,
        bool consumeBatches, nsecs_t frameTime) {
  env->CallVoidMethod(mReceiverObjGlobal,
                      gInputEventReceiverClassInfo.dispatchInputEvent, seq, inputEventObj);
}
```

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

**[cpp]**

```cpp
private void dispatchInputEvent(int seq, InputEvent event) {
        mSeqMap.put(event.getSequenceNumber(), seq);
        onInputEvent(event);
}
```

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

```
[cpp]
final class WindowInputEventReceiver extends InputEventReceiver {
  public WindowInputEventReceiver(InputChannel inputChannel, Looper looper) {
    super(inputChannel, looper);
  }
  @Override
  public void onInputEvent(InputEvent event) {
    enqueueInputEvent(event, this, 0, true);
  }
}
void enqueueInputEvent(InputEvent event,
    InputEventReceiver receiver, int flags, boolean processImmediately) {
  scheduleProcessInputEvents();
}
```

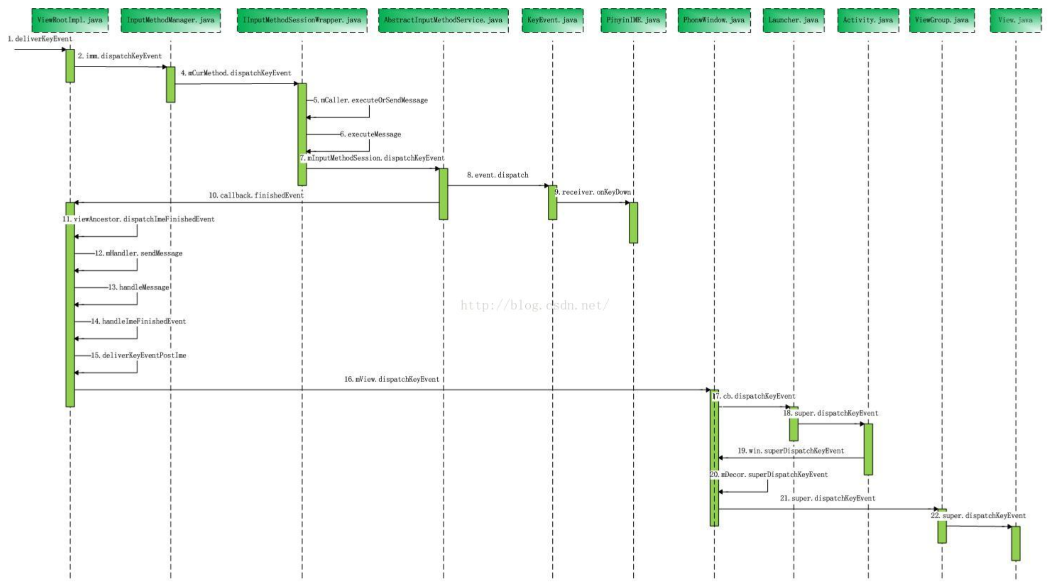
///////////////////////////////////////////////////////////////////////////

有关handler机制请看下文：

http://blog.csdn.net/itachi85/article/details/8035333

```
[cpp]
final ViewRootHandler mHandler = new ViewRootHandler();
private void scheduleProcessInputEvents() {
  Message msg = mHandler.obtainMessage(MSG_PROCESS_INPUT_EVENTS);
  mHandler.sendMessage(msg);
}
public void handleMessage(Message msg) {
  switch (msg.what) {
    case MSG_PROCESS_INPUT_EVENTS:
      doProcessInputEvents();
  }
}
```

////////////////////////////////////////////////////////////////////


ViewRootImpl.java
InputMethodManager.java
IInputMethodSessionWrapper.java
AbstractInputMethodService.java
KeyEvent.java
PinyinIME.java
PhonwWindow.java
Launcher.java
Activity.java
ViewGroup.java
View.java
1. deliverKeyEvent
2. imm.dispatchKeyEvent
4. mCurMethod.dispatchKeyEvent
5. mCaller.executeOrSendMessage
6. executeMessage
7. mInputMethodSession.dispatchKeyEvent
8. event.dispatch
9. receiver.onKeyDown
10. callback.finishedEvent
11. viewAncestor.dispatchImeFinishedEvent
12. mHandler.sendMessage
13. handleMessage
14. handleImeFinishedEvent
15. deliverKeyEventPostIme
16. mView.dispatchKeyEvent
17. cb.dispatchKeyEvent
18. super.dispatchKeyEvent
19. win.superDispatchKeyEvent
20. mDecor.superDispatchKeyEvent
21. super.dispatchKeyEvent
22. super.dispatchKeyEvent
http://blog.csdn.net/

**这其中ViewRootImpl.java的deliverKeyEventPostIme接口中在调用mView.dispatchKeyEvent(event)返回为false时，会再次调用mFallbackEventHandler.dispatchKeyEvent(event)让系统做默认处理。**

```
[cpp]
void doProcessInputEvents() {
  deliverInputEvent(q);
}
private void deliverInputEvent(QueuedInputEvent q) {
  deliverKeyEvent(q);
  deliverPointerEvent(q);
  deliverTrackballEvent(q);
  deliverGenericMotionEvent(q);
}
private void deliverKeyEvent(QueuedInputEvent q) {
  imm.dispatchKeyEvent(mView.getContext(), seq, event, mInputMethodCallback); //分发给输入法
  deliverKeyEventPostIme(q);//分发给View
  /*
  private void deliverKeyEventPostIme(QueuedInputEvent q) {
```

```java
    mView.dispatchKeyEvent(event)
  }
  */
}
private void deliverPointerEvent(QueuedInputEvent q) {
  boolean handled = mView.dispatchPointerEvent(event); //分发给View
}
private void deliverTrackballEvent(QueuedInputEvent q) {
  imm.dispatchTrackballEvent(mView.getContext(), seq, event,
    mInputMethodCallback);  //分发给输入法
  deliverTrackballEventPostIme(q);  //分发给View
  /*
  private void deliverTrackballEventPostIme(QueuedInputEvent q) {
    mView.dispatchTrackballEvent(event)
  }
  */
}
private void deliverGenericMotionEvent(QueuedInputEvent q) {
  imm.dispatchGenericMotionEvent(mView.getContext(), seq, event,
    mInputMethodCallback);  //分发给输入法
  deliverGenericMotionEventPostIme(q); //分发给View
  /*
  private void deliverGenericMotionEventPostIme(QueuedInputEvent q) {
    updateJoystickDirection(event, false); //游戏手柄的摇杆就是在这处理
    mView.dispatchGenericMotionEvent(event)
  }
  */
}
```

分发给应用程序Activity：

frameworks/base/policy/src/com/android/internal/policy/impl/PhoneWindow.java

[java]

```java
private final class DecorView extends FrameLayout implements RootViewSurfaceTaker {
  public boolean dispatchKeyEvent(KeyEvent event) {
    final Callback cb = getCallback();
    //cb为应用程序MainActivity
    final boolean handled = cb != null && mFeatureId < 0 ? cb.dispatchKeyEvent(event) : super.dispatchKeyEvent(event);
    //给应用程序Activity的dispatchKeyEvent处理或交给View的dispatchKeyEvent
  }
}
```

而上述应用程序中的dispatchKeyEvent一般会调用其父类的该方法，例如：

packages/apps/Launcher2/src/com/android/launcher2/Launcher.java

[java]

```java
public boolean dispatchKeyEvent(KeyEvent event) {
  return super.dispatchKeyEvent(event);
}
```

应用程序Activity在分发给与之关联的某个View，如果这个View没有处理、最终交给该Activity自己处理。

应用程序有关View的设置：

[java]

```java
private Dialog mMenuWin;
mMenuWin = new Dialog(aActivity, R.style.CameraDialog);
mMenuWin.setContentView(mMenuLayout);
mMenuWin.setOnClickListener();  //鼠标单击
mMenuWin.setOnLongClickListener();  //
mMenuWin.setOnTouchListener(); //触摸板
mMenuWin.setOnKeyListener(new OnKeyListener() {
  public boolean onKey();  //按键
  public void onClick(View v); //鼠标单击
}
```

frameworks/base/core/java/android/app/Activity.java

[java]

```java
public boolean dispatchKeyEvent(KeyEvent event) {
  onUserInteraction();
  Window win = getWindow();
  if (win.superDispatchKeyEvent(event)) { //首先由Window消化，即如果View消化了、则Activity将不在回调onKeyDown

    return true;
  }
  View decor = mDecor; //如果没被消化，会调用Activity的onKeyDown
  if (decor == null) decor = win.getDecorView();
```

```java
    return event.dispatch(this, decor != null ? decor.getKeyDispatcherState() : null, this);
  }
}
```

我们重点分析win.superDispatchKeyEvent，也就是View的处理流程：
frameworks/base/policy/src/com/android/internal/policy/impl/PhoneWindow.java

```java
[java]
public class PhoneWindow extends Window implements MenuBuilder.Callback {
  public boolean superDispatchKeyEvent(KeyEvent event) {
    return mDecor.superDispatchKeyEvent(event);
  }
}
private final class DecorView extends FrameLayout implements RootViewSurfaceTaker {
  public boolean superDispatchKeyEvent(KeyEvent event) {
    super.dispatchKeyEvent(event)
  }
}
```

frameworks/base/core/java/android/view/ViewGroup.java //分发给View的关键部分！！！

```java
[java]
public boolean dispatchKeyEvent(KeyEvent event) {
  mInputEventConsistencyVerifier.onKeyEvent(event, 1);
  super.dispatchKeyEvent(event)
}
```

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

```java
[java]
public boolean dispatchKeyEvent(KeyEvent event) {
  li.mOnKeyListener.onKey(this, event.getKeyCode(), event);  //回调应用程序View相应方法
  event.dispatch(this, mAttachInfo != null ? mAttachInfo.mKeyDispatchState : null, this)
  /*
  frameworks/base/core/java/android/view/KeyEvent.java
  public final boolean dispatch(Callback receiver, DispatcherState state,
    Object target) {
    //按键响应
    boolean res = receiver.onKeyDown(mKeyCode, this); //应用程序回调函数
  }
  */
}
public final boolean dispatchPointerEvent(MotionEvent event) {
  if (event.isTouchEvent()) {
    return dispatchTouchEvent(event);
  } else {
    return dispatchGenericMotionEvent(event);
  }
}
public boolean dispatchTouchEvent(MotionEvent event) {
  //触摸板响应
  li.mOnTouchListener.onTouch(this, event) //应用程序继承OnTouchListener，实现的回调接口
  //鼠标左键响应
  onTouchEvent(event)
  /*
  public boolean onTouchEvent(MotionEvent event) {
    performClick();
    //该接口调用li.mOnClickListener.onClick(this);为应用程序继承OnClickListener的回调函数
  }
  */
}
```

以下不再做分析
dispatchGenericMotionEvent
dispatchTrackballEvent

dispatchConfigurationChanged //添加或删除键盘设备Activity重启，见http://blog.csdn.net/tankai19880619/article/details/16805401

三、Input设备与Activity关系

1.InputReaderThread线程检测到设备插入删除

frameworks/base/service/input/InputReader.cpp

```cpp
[cpp]
void InputReader::loopOnce() {
  size_t count = mEventHub->getEvents(timeoutMillis, mEventBuffer, EVENT_BUFFER_SIZE);
  /*
  frameworks/base/services/input/EventHub.cpp
  size_t EventHub::getEvents(int timeoutMillis, RawEvent* buffer, size_t bufferSize) {
    int32_t readSize = read(device->fd, readBuffer,
      sizeof(struct input_event) * capacity);//从驱动读取事件
  }
  */
  processEventsLocked(mEventBuffer, count);
}
void InputReader::processEventsLocked(const RawEvent* rawEvents, size_t count) {
  case EventHubInterface::FINISHED_DEVICE_SCAN:
    handleConfigurationChangedLocked(rawEvent->when);
}
void InputReader::handleConfigurationChangedLocked(nsecs_t when) {
  updateGlobalMetaStateLocked();
  // Enqueue configuration changed.
  NotifyConfigurationChangedArgs args(when);
  mQueuedListener->notifyConfigurationChanged(&args);
}
```

说明：有的平台需要在接入硬件键盘时Activity不需要刷新；可以在上处做屏蔽：

[cpp]

```cpp
// add by tank
// do not send configuration change
//NotifyConfigurationChangedArgs args(when);
//mQueuedListener->notifyConfigurationChanged(&args);
// end tank
```

2.InputReaderThread线程分发给InputDispatcherThread线程

frameworks/base/service/input/InputDispatcher.cpp

[cpp]

```cpp
void InputDispatcher::notifyConfigurationChanged(const NotifyConfigurationChangedArgs* args) {
  needWake = enqueueInboundEventLocked(newEntry);
  if (needWake) {
    mLooper->wake();
  }
}
```

3.InputReaderThread线程收到消息并处理
frameworks/base/service/input/InputDispatcher.cpp

[cpp]

```cpp
bool InputDispatcherThread::threadLoop() {
  mDispatcher->dispatchOnce();
  return true;
}
void InputDispatcher::dispatchOnce() {
  dispatchOnceInnerLocked(&nextWakeupTime);
}
void InputDispatcher::dispatchOnceInnerLocked(nsecs_t* nextWakeupTime) {
  case EventEntry::TYPE_CONFIGURATION_CHANGED: {
    ConfigurationChangedEntry* typedEntry =
            static_cast<ConfigurationChangedEntry*>(mPendingEvent);
  done = dispatchConfigurationChangedLocked(currentTime, typedEntry);
  }
}
bool InputDispatcher::dispatchConfigurationChangedLocked(
        nsecs_t currentTime, ConfigurationChangedEntry* entry) {
  CommandEntry* commandEntry = postCommandLocked(
          & InputDispatcher::doNotifyConfigurationChangedInterruptible);
}
void InputDispatcher::doNotifyConfigurationChangedInterruptible(
        CommandEntry* commandEntry) {
  mPolicy->notifyConfigurationChanged(commandEntry->eventTime);
}
```

如上，不再做分析：
frameworks/base/services/jni/com_android_server_input_InputManagerService.cpp

[cpp]

```cpp
void NativeInputManager::notifyConfigurationChanged(nsecs_t when) {
  env->CallVoidMethod(mServiceObj, gServiceClassInfo.notifyConfigurationChanged, when);
}
```

frameworks/base/services/java/com/android/server/input/InputManagerService.cpp

[cpp]

```cpp
private void notifyConfigurationChanged(long whenNanos) {
  mWindowManagerCallbacks.notifyConfigurationChanged();
}
```

如上，不再做分析：
frameworks/base/service/java/com/android/server/wm/InputMonitor.java

[cpp]
```
public void notifyConfigurationChanged() {
  mService.sendNewConfiguration();
}
```

frameworks/base/service/java/com/android/server/wm/WindowManagerService.java

[cpp]
```
void sendNewConfiguration() {
  mActivityManager.updateConfiguration(null);
  /*
  mActivityManager = ActivityManagerNative.getDefault();
  frameworks/base/core/java/android/app/ActivityManagerNative.java
  static public IActivityManager getDefault() {
    return gDefault.get();
  }
  private static final Singleton<IActivityManager> gDefault = new Singleton<IActivityManager>() {
    IBinder b = ServiceManager.getService("activity");
    IActivityManager am = asInterface(b);
    return am;
  }
  frameworks/base/services/java/com/android/server/am/ActivityManagerService.java
  public static void setSystemProcess() {
    ActivityManagerService m = mSelf;
    ServiceManager.addService("activity", m, true);
  }
  */
}
```

4.交由ActivityManagerService进程处理
frameworks/base/services/java/com/android/server/am/ActivityManagerService.java

[cpp]
```
public void updateConfiguration(Configuration values) {
  updateConfigurationLocked(values, null, false, false);
}
boolean updateConfigurationLocked(Configuration values,
            ActivityRecord starting, boolean persistent, boolean initLocale) {
  kept = mMainStack.ensureActivityConfigurationLocked(starting, changes);
  public void setWindowManager(WindowManagerService wm) {
    mWindowManager = wm;
  }
}
```

frameworks/base/services/java/com/android/server/am/ActivityStack.java

[cpp]
```
final boolean ensureActivityConfigurationLocked(ActivityRecord r,
            int globalChanges) {
  //一般会重启Activity
  if ((changes&(~r.info.getRealConfigChanged())) != 0 || r.forceNewConfig) {
    relaunchActivityLocked(r, r.configChangeFlags, false);
    return false;
  }
  //应用程序AndroidMenifest中写标记将不会重启
  r.app.thread.scheduleActivityConfigurationChanged(r.appToken);
}
```

frameworks/base/core/java/android/app/ActivityThread.java

[cpp]
```
public void scheduleActivityConfigurationChanged(IBinder token) {
  queueOrSendMessage(H.ACTIVITY_CONFIGURATION_CHANGED, token);
}
//消息循环同上，不再分析
public void handleMessage(Message msg) {
  case ACTIVITY_CONFIGURATION_CHANGED:
    handleActivityConfigurationChanged((IBinder)msg.obj);
}
final void handleActivityConfigurationChanged(IBinder token) {
  performConfigurationChanged(r.activity, mCompatConfiguration);
}
private static void performConfigurationChanged(ComponentCallbacks2 cb, Configuration config) {
  cb.onConfigurationChanged(config); //回调Activity类的onConfigurationChanged方法
}
```

## 四、项目问题

### 1.resumeTopActivity时的Activity重启。

http://blog.csdn.net/jivin_shen/article/details/6839175

3

操作逻辑：打开Launcher界面下的一个应用（比如播放器），完后接入USB键盘；之后退出该应用，也就是resumeTopActivity到Launcher时也引发了config配置更新导致的Activity重启。

原理以及解决部分：

rameworks/base/services/java/com/android/server/am/ActivityStack.java

```cpp
[cpp]
final boolean resumeTopActivityLocked(ActivityRecord prev) {
  return resumeTopActivityLocked(prev, null);
}
final boolean resumeTopActivityLocked(ActivityRecord prev, Bundle options) {
  Configuration config = mService.mWindowManager.updateOrientationFromAppTokens(
                          mService.mConfiguration,
                          next.mayFreezeScreenLocked(next.app) ? next.appToken : null);
}
```

frameworks/base/services/java/com/android/server/wm/WindowManagerService.java

```cpp
[cpp]
public Configuration updateOrientationFromAppTokens(
            Configuration currentConfig, IBinder freezeThisOneIfNeeded) {
  config = updateOrientationFromAppTokensLocked(currentConfig,
                  freezeThisOneIfNeeded);
}
private Configuration updateOrientationFromAppTokensLocked(
            Configuration currentConfig, IBinder freezeThisOneIfNeeded) {
  computeScreenConfigurationLocked(mTempConfiguration)
}
boolean computeScreenConfigurationLocked(Configuration config) {
  if ((sources & InputDevice.SOURCE_TOUCHSCREEN) == InputDevice.SOURCE_TOUCHSCREEN) {
    //change by tank
    config.touchscreen = Configuration.TOUCHSCREEN_NOTOUCH;
    //config.touchscreen = Configuration.TOUCHSCREEN_FINGER;
    //end tank
  }
  else if ((sources & InputDevice.SOURCE_DPAD) == InputDevice.SOURCE_DPAD
                          && config.navigation == Configuration.NAVIGATION_NONAV) {
    //change by tank
    //config.navigation = Configuration.NAVIGATION_DPAD;
    //navigationPresence |= presenceFlag;
    //end tank
  }
  if (device.getKeyboardType() == InputDevice.KEYBOARD_TYPE_ALPHABETIC) {
    //change by tank
    //config.keyboard = Configuration.KEYBOARD_QWERTY;
    //keyboardPresence |= presenceFlag;
    //end tank
  }
}
```

### **2.面板设备与虚拟驱动导致的up上报两次：**

drop类按键

down或up：

dispatchOnceInnerLocked>

dropInboundEventLocked>synthesizeCancelationEventsForAllConnectionsLocked-synthesizeCancelationEventsForConnectionLocked>inputState.synthesizeCancelationEvents->mKeyMementos.itemAt(i)，最后上报系统（synthesizeCancelationEventsForConnectionLocked调用enqueueDispatchEntryLocked）

非drop类按键

down：

dispatchOnceInnerLocked->

dispatchKeyLocked->dispatchEventLocked->prepareDispatchCycleLocked->enqueueDispatchEntriesLocked->enqueueDispatchEntryLocked->InputState::trackKey->addKeyMemento //只在down时保存对up的处理

问题：

面板down->drop

虚拟down->非drop，保存up

面板down->drop，将虚拟保存的up送上去

虚拟up->非drop，直接上报

结果——两个虚拟的up

修改方法：

frameworks/base/service/input/InputDispatcher.cpp

```cpp
[cpp]
void InputDispatcher::enqueueDispatchEntryLocked(
        const sp<Connection>& connection, EventEntry* eventEntry, const InputTarget* inputTarget,
        int32_t dispatchMode)
{
  if (!connection->inputState.trackKey(keyEntry,
                dispatchEntry->resolvedAction, dispatchEntry->resolvedFlags)/*add by tank@tcl.com end */ |
| (dispatchEntry->resolvedFlags == 0x28))
  {
    #if DEBUG_DISPATCH_CYCLE
    ALOGD("channel '%s' ~ enqueueDispatchEntryLocked: skipping inconsistent key event",
                connection->getInputChannelName());
    #endif
    delete dispatchEntry;
    return; // skip the inconsistent event
  }
  /*
  //add by tankai
  if(dispatchEntry->resolvedFlags == 0x28 && keyEntry->deviceId == 3){
    ALOGD("TK--------->>delete sim KeyMementos up\n");
    delete dispatchEntry;
    return; // skip the inconsistent event
  }
  //end tankai
  */
}
```

3.焦点request错误导致不能响应按键

正确调用：setFocusable(true)和requestFocus()重新获取焦点

错误调用：setFocusable(false)和requestFocus()

系统侧为该应用tv.huan.deezer强制修改：

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

```java
[java]
public final boolean requestFocus() {
        Log.d("TKTK","TK---->>View.java>>>requestFocus()");//add by tank
        if(SystemProperties.get("sys.user.camera",null).equals("tv.huan.deezer"))
        {
          setFocusable(true);
        }
        //end tank
        return requestFocus(View.FOCUS_DOWN);
    }
```

该文章已被禁止评论！

**Android Framework------之Input子系统** wangkaiblog 2013-09-27 11:25:05 11126

http://www.cnblogs.com/haiming/p/3318614.html 下面这是基于Android4.2代码的关于Input子系统的笔记。在这篇笔记中，只涉及Android相...

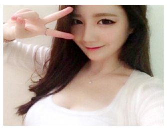
3

**Linux/Android——input子系统核心 (三)** jscese 2014-12-26 15:10:07 5103

前的博客有涉及到linux的input子系统，这里学习记录一下input模块. input子系统，作为管理输入设备与系统进行交互的中，任何的输入设备驱动都要通过input向内核注册其设备， 常...

**程序员不会英语怎么行？**

老司机教你一个数学公式秒懂天下英语



广告

**input子系统整体框架** zjngogo 2015-06-04 16:24:24 1488

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 ），驱动层和事件处理层（Event Handler）三部份组成。Android层操作input子...

**android input子系统之三：事件层** lixuehui848 2016-01-22 16:04:25 477

四 事件层 struct input_dev物理输入设备的基本数据结构,包含设备相关的一些信息 structinput_handler 事件处理结构体,定义怎么处理事件的逻辑 struct in...

**呼叫中心系统**

专业的呼叫中心系统

百度广告

**android input系统** bsxiaomage 2015-04-06 21:50:50 1302

linux内核的input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，触摸屏，加速计和手写板)等字符设备进行统一处理的一层抽象，就是在字符设备驱动上抽象出的一层。input...

**Andriod Input子系统框架** hongwazi_2010 2015-01-25 20:27:09 902

原文地址：http://www.cnblogs.com/haiming/p/3318614.html 下面这是基于Android4.2代码的关于Input子系统的笔记。在这篇笔记中，只涉及And...

**Android Input子系统浅谈** tiantangniaochao 2016-01-11 15:30:34 504

Android Input子系统浅谈本文主要讲解[Android Input 子系统][6]，我会从一下几个方面讲解： linux kernel的input子系统框架 以触摸屏驱动为例讲解内核inpu...

**input输入子系统整体流程** mike8825 2016-03-02 19:37:25 1644

input输入子系统整体流程 本节分析input子系统在内核中的实现，包括输入子系统（Input Core），事件处理层（Event Handler）和设备驱动层。由于上节代码讲解了设备驱动层的写法...

**《Android系统学习》第一章：Input子系统驱动部分**

==========================================================================================...

tankai19880619 2012-10-24 18:03:13 1323

**熟悉笔记—数据结构（c语言版） 之 顺序表** maimang1001 2011-01-02 22:33:00 341

#include typedef int ElemType; #define INITSIZE 100  typedef struct {    ElemType *data; ...

**核中的锁机制** ruanjianruanjianruan 2017-02-11 23:00:19 515

3 atomic(原子操作): 原型：atomic_t数据类型，atomic_inc(atomic_t *v)将v加1 1，原子操作是不可分割的，在执行完毕不会被任何其它任务或事件中断。 在单处理器系统(...

**ndroid4.0 input子系统分析(kernel部分)** lin364812726 2014-04-26 16:32:08 385

一、前言 前面我们分析了android的input子系统的android部分的代码，下面我们继续来分析kernel部分的，对于这个系统kernel部分和标准linux差别不大， goo...

**Linux/Android——输入子系统input_event传递 (二)**

版权声明：免责声明： 本人在此发文（包括但不限于汉字、拼音、拉丁字母）均为随意敲击键盘所出，用于检验本人电脑键盘录入、屏幕显示的机械、光电性能，并不代表本人局部或全部同意、支持或者反对观点。如需要详查...

u013491946 2017-05-23 13:28:12 227

**Android4.2 Input子系统** tankai19880619 2012-12-03 17:39:14 3202

==================================================================================================...

**android Input子系统分析** xiaoxiaoyu1107 2014-09-19 16:39:06 528

Input Technical Information Android 输入子系统支持许多不同的设备类，包括键盘，摇杆，轨迹球，鼠标和触摸屏. 这份文档描述了上层如何配置，校...

**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Android 5.0中事件输入子系统的框架和流程没有本质变化。Service...

thinkinwm 2015-10-19 21:53:38 1752

**Android 4.2 Input Event事件处理流程<—>---应用注册**

一个应用要接受Android的各种input消息，就需要将自己注册进去，这样底层收到消息后...

new_abc 2014-07-30 19:37:37 6227

**Android 触摸消息处理** Siobhan 2012-12-05 10:58:35 11487

1. WindowInputEventReceiver.onInputEvent() ----ViewRootImpl.java 从InputDispatch中publish一个Eent...

**Android InputEvent(Motion/Key/Sleep....) 流程跟踪**

ViewRootImpl的setView()函数中（这个函数被调用代表着Activity的界面基本建立）会建立一群InputStage并以职责链模式链接起来进行协同工作：syntheticInputS...

fyfcauc 2015-11-20 16:18:36 1244

**Android输入事件从读取到分发五：事件分发前的拦截过程**

在前面的文章：Android输入事件从读取到分发三：InputDispatcherThread线程分发事件的过程 一文中已经提过事件在分发前要做拦截的事情，只不过当时没有展开来分析，因此这篇文章的主要...

u011913612 2016-11-07 17:26:18 5423

3