原 Android Input流程分析（四）：InputDispatcher

2017年12月20日 02:06:08 Invoker123 阅读数：657

之前提到，InputReader将Key的信息封装成一个NotifyKeyArgs对象，调用InputDispatcher的notifyKey来处理。再进行一些初步的处 将个NotifyKeyArgs对象拆包，得到的信息构成一个KeyEvent，并调用interceptKeyBeforeQueueing进行第一次事件拦截，该函数最终会调到java层PhoneWir Manager的interceptKeyBeforeQueueing函数，对一些系统特殊按键进行处理。该函数会返回一个result，如果result带有ACTION_PASS_TO_USER标 说明在这次的事件拦截中没有被过滤；如果没有ACTION_PASS_TO_USER标志位，则说明按键在这次事件拦截中被过滤处理掉了。这个ACTION_PASS_TO_US 志位将被记录在Native层interceptKeyBeforeQueueing函数第二个参数policyFlags中。接下来构造一个KeyEntry，将这个KeyEntry通过enqueueInbound Locked加入到mInboundQueue这个Queue< EventEntry>中。每次InputReader推送NotifyKeyArgs对象过来而且mInboundQueue为空时，就会调用wake函数唤醒mLooper，wake函数就是往mWakeEventFd表示的fd写入一个数字进行唤醒。

/frameworks/native/services/inputflinger/InputDispatcher.cpp

```
void InputDispatcher::notifyKey(const NotifyKeyArgs* args) {
#if DEBUG_INBOUND_EVENT_DETAILS
    ALOGD("notifyKey - eventTime=%lld, deviceId=%d, source=0x%x, policyFlags=0x%x, action=0x%x, "
            "flags=0x%x, keyCode=0x%x, scanCode=0x%x, metaState=0x%x, downTime=%lld",
            args->eventTime, args->deviceId, args->source, args->policyFlags,
            args->action, args->flags, args->keyCode, args->scanCode,
            args->metaState, args->downTime);
#endif
    if (!validateKeyEvent(args->action)) {
        return;
    }

    uint32_t policyFlags = args->policyFlags;
    int32_t flags = args->flags;
    int32_t metaState = args->metaState;
    if ((policyFlags & POLICY_FLAG_VIRTUAL) || (flags & AKEY_EVENT_FLAG_VIRTUAL_HARD_KEY)) {
        policyFlags |= POLICY_FLAG_VIRTUAL;
        flags |= AKEY_EVENT_FLAG_VIRTUAL_HARD_KEY;
    }
    if (policyFlags & POLICY_FLAG_FUNCTION) {
        metaState |= AMETA_FUNCTION_ON;
    }

    policyFlags |= POLICY_FLAG_TRUSTED;

    int32_t keyCode = args->keyCode;
    if (metaState & AMETA_META_ON && args->action == AKEY_EVENT_ACTION_DOWN) {
        int32_t newKeyCode = AKEYCODE_UNKNOWN;
        if (keyCode == AKEYCODE_DEL) {
            newKeyCode = AKEYCODE_BACK;
        } else if (keyCode == AKEYCODE_ENTER) {
            newKeyCode = AKEYCODE_HOME;
        }
        if (newKeyCode != AKEYCODE_UNKNOWN) {
            AutoMutex _l(mLock);
            struct KeyReplacement replacement = {keyCode, args->deviceId};
            mReplacedKeys.add(replacement, newKeyCode);
            keyCode = newKeyCode;
            metaState &= ~AMETA_META_ON;
        }
    } else if (args->action == AKEY_EVENT_ACTION_UP) {
        // In order to maintain a consistent stream of up and down events, check to see if the key
        // going up is one we've replaced in a down event and haven't yet replaced in an up event,
        // even if the modifier was released between the down and the up events.
        AutoMutex _l(mLock);
        struct KeyReplacement replacement = {keyCode, args->deviceId};
        ssize_t index = mReplacedKeys.indexOfKey(replacement);
        if (index >= 0) {
            keyCode = mReplacedKeys.valueAt(index);
            mReplacedKeys.removeItemsAt(index);
            metaState &= ~AMETA_META_ON;
        }
    }

    KeyEvent event;
    event.initialize(args->deviceId, args->source, args->action,
```

```
    mPolicy->interceptKeyBeforeQueueing(&event, /*byref*/ policyFlags);

    bool needWake;
    { // acquire lock
        mLock.lock();

        if (shouldSendKeyToInputFilterLocked(args)) {
            mLock.unlock();

            policyFlags |= POLICY_FLAG_FILTERED;
            if (!mPolicy->filterInputEvent(&event, policyFlags)) {
                return; // event was consumed by the filter
            }

            mLock.lock();
        }

        int32_t repeatCount = 0;
        KeyEntry* newEntry = new KeyEntry(args->eventTime,
                args->deviceId, args->source, policyFlags,
                args->action, flags, keyCode, args->scanCode,
                metaState, repeatCount, args->downTime);

        needWake = enqueueInboundEventLocked(newEntry);
        mLock.unlock();
    } // release lock

    if (needWake) {
        mLooper->wake();
    }
}
```

InputDispatcher是在一个threadLoop循环中执行dispatchOnce函数的。haveCommandsLocked判断mCommandQueue是否不为空。这里假设是在首次执行dispatchOnce函数时，mCommandQueue理应为空，dispatchOnceInnerLocked函数会被执行。

/frameworks/native/services/inputflinger/InputDispatcher.cpp

```
void InputDispatcher::dispatchOnce() {
    nsecs_t nextWakeupTime = LONG_LONG_MAX;
    { // acquire lock
        AutoMutex _l(mLock);
        mDispatcherIsAliveCondition.broadcast();

        // Run a dispatch loop if there are no pending commands.
        // The dispatch loop might enqueue commands to run afterwards.
        if (!haveCommandsLocked()) {
            dispatchOnceInnerLocked(&nextWakeupTime);
        }

        // Run all pending commands if there are any.
        // If any commands were run then force the next poll to wake up immediately.
        if (runCommandsLockedInterruptible()) {
            nextWakeupTime = LONG_LONG_MIN;
        }
    } // release lock

    // Wait for callback or timeout or wake.  (make sure we round up, not down)
    nsecs_t currentTime = now();
    int timeoutMillis = toMillisecondTimeoutDelay(currentTime, nextWakeupTime);
    mLooper->pollOnce(timeoutMillis);
}
```

这里会取出mInboundQueue队列头的KeyEntry作为mPendingEvent继续往后处理。随后通过pokeUserActivityLocked将InputDispatcher::doPokeUserActivityLockedInterruptible函数推入到mCommandQueue里面。

接下来根据情况设置按键丢弃标志。凡是在第一次事件拦截中没有被加上POLICY_FLAG_PASS_TO_USER的Key，其dropReason会被设为DROP_REASON_POLICY。还有一些规则可以细看一下。

/frameworks/native/services/inputflinger/InputDispatcher.cpp

```
void InputDispatcher::dispatchOnceInnerLocked(nsecs_t* nextWakeupTime) {
    nsecs_t currentTime = now();

    // Reset the key repeat timer whenever normal dispatch is suspended while the
```

```
        resetKeyRepeatLocked();
    }

    // If dispatching is frozen, do not process timeouts or try to deliver any new events.
    if (mDispatchFrozen) {
#if DEBUG_FOCUS
        ALOGD("Dispatch frozen.  Waiting some more.");
#endif
        return;
    }

    // Optimize latency of app switches.
    // Essentially we start a short timeout when an app switch key (HOME / ENDCALL) has
    // been pressed.  When it expires, we preempt dispatch and drop all other pending events.
    bool isAppSwitchDue = mAppSwitchDueTime <= currentTime;
    if (mAppSwitchDueTime < *nextWakeupTime) {
        *nextWakeupTime = mAppSwitchDueTime;
    }

    // Ready to start a new event.
    // If we don't already have a pending event, go grab one.
    if (! mPendingEvent) {
        if (mInboundQueue.isEmpty()) {
            if (isAppSwitchDue) {
                // The inbound queue is empty so the app switch key we were waiting
                // for will never arrive.  Stop waiting for it.
                resetPendingAppSwitchLocked(false);
                isAppSwitchDue = false;
            }

            // Synthesize a key repeat if appropriate.
            if (mKeyRepeatState.lastKeyEntry) {
                if (currentTime >= mKeyRepeatState.nextRepeatTime) {
                    mPendingEvent = synthesizeKeyRepeatLocked(currentTime);
                } else {
                    if (mKeyRepeatState.nextRepeatTime < *nextWakeupTime) {
                        *nextWakeupTime = mKeyRepeatState.nextRepeatTime;
                    }
                }
            }

            // Nothing to do if there is no pending event.
            if (!mPendingEvent) {
                return;
            }
        } else {
            // Inbound queue has at least one entry.
            mPendingEvent = mInboundQueue.dequeueAtHead();
            traceInboundQueueLengthLocked();
        }

        // Poke user activity for this event.
        if (mPendingEvent->policyFlags & POLICY_FLAG_PASS_TO_USER) {
            pokeUserActivityLocked(mPendingEvent);
        }

        // Get ready to dispatch the event.
        resetANRTimeoutsLocked();
    }

    // Now we have an event to dispatch.
    // All events are eventually dequeued and processed this way, even if we intend to drop them.
    ALOG_ASSERT(mPendingEvent != NULL);
    bool done = false;
    DropReason dropReason = DROP_REASON_NOT_DROPPED;
    if (!(mPendingEvent->policyFlags & POLICY_FLAG_PASS_TO_USER)) {
        dropReason = DROP_REASON_POLICY;
    } else if (!mDispatchEnabled) {
        dropReason = DROP_REASON_DISABLED;
    }

    if (mNextUnblockedEvent == mPendingEvent) {
        mNextUnblockedEvent = NULL;
    }

    switch (mPendingEvent->type) {
```

```
                static_cast<ConfigurationChangedEntry*>(mPendingEvent);
        done = dispatchConfigurationChangedLocked(currentTime, typedEntry);
        dropReason = DROP_REASON_NOT_DROPPED; // configuration changes are never dropped
        break;
    }

    case EventEntry::TYPE_DEVICE_RESET: {
        DeviceResetEntry* typedEntry =
                static_cast<DeviceResetEntry*>(mPendingEvent);
        done = dispatchDeviceResetLocked(currentTime, typedEntry);
        dropReason = DROP_REASON_NOT_DROPPED; // device resets are never dropped
        break;
    }

    case EventEntry::TYPE_KEY: {
        KeyEntry* typedEntry = static_cast<KeyEntry*>(mPendingEvent);
        if (isAppSwitchDue) {
            if (isAppSwitchKeyEventLocked(typedEntry)) {
                resetPendingAppSwitchLocked(true);
                isAppSwitchDue = false;
            } else if (dropReason == DROP_REASON_NOT_DROPPED) {
                dropReason = DROP_REASON_APP_SWITCH;
            }
        }
        if (dropReason == DROP_REASON_NOT_DROPPED
                && isStaleEventLocked(currentTime, typedEntry)) {
            dropReason = DROP_REASON_STALE;
        }
        if (dropReason == DROP_REASON_NOT_DROPPED && mNextUnblockedEvent) {
            dropReason = DROP_REASON_BLOCKED;
        }
        done = dispatchKeyLocked(currentTime, typedEntry, &dropReason, nextWakeupTime);
        break;
    }

    case EventEntry::TYPE_MOTION: {
        MotionEntry* typedEntry = static_cast<MotionEntry*>(mPendingEvent);
        if (dropReason == DROP_REASON_NOT_DROPPED && isAppSwitchDue) {
            dropReason = DROP_REASON_APP_SWITCH;
        }
        if (dropReason == DROP_REASON_NOT_DROPPED
                && isStaleEventLocked(currentTime, typedEntry)) {
            dropReason = DROP_REASON_STALE;
        }
        if (dropReason == DROP_REASON_NOT_DROPPED && mNextUnblockedEvent) {
            dropReason = DROP_REASON_BLOCKED;
        }
        done = dispatchMotionLocked(currentTime, typedEntry,
                &dropReason, nextWakeupTime);
        break;
    }

    default:
        ALOG_ASSERT(false);
        break;
    }

    if (done) {
        if (dropReason != DROP_REASON_NOT_DROPPED) {
            dropInboundEventLocked(mPendingEvent, dropReason);
        }
        mLastDropReason = dropReason;

        releasePendingEventLocked();
        *nextWakeupTime = LONG_LONG_MIN;  // force next poll to wake up immediately
    }
}
```

不管是会被丢弃的按键还是保留的按键，都会进入到dispatchKeyLocked函数中。前面是生成重复按键事件，在 mConfig.keyRepeatTimeout时间内同一个按键被按下会被视为重复按键事件，即是长按事件，通过其repeatCount成员是否大于1体现。接着会将doInterceptKeyBeforeDispatchingLockedInterruptible推入到mCommandQueue中，这是第二次执行时间拦截的函数。findFocusedWindowTargetsLocked接受当前焦点窗口句柄mFocusedWindowHandle为参数，构造出一个InputTarget，push到inputTargets中去。

/frameworks/native/services/inputflinger/InputDispatcher.cpp

```
    // Preprocessing.
    if (! entry->dispatchInProgress) {
        if (entry->repeatCount == 0
                && entry->action == AKEY_EVENT_ACTION_DOWN
                && (entry->policyFlags & POLICY_FLAG_TRUSTED)
                && (!(entry->policyFlags & POLICY_FLAG_DISABLE_KEY_REPEAT))) {
            if (mKeyRepeatState.lastKeyEntry
                    && mKeyRepeatState.lastKeyEntry->keyCode == entry->keyCode) {
                // We have seen two identical key downs in a row which indicates that the device
                // driver is automatically generating key repeats itself.  We take note of the
                // repeat here, but we disable our own next key repeat timer since it is clear that
                // we will not need to synthesize key repeats ourselves.
                entry->repeatCount = mKeyRepeatState.lastKeyEntry->repeatCount + 1;
                resetKeyRepeatLocked();
                mKeyRepeatState.nextRepeatTime = LONG_LONG_MAX; // don't generate repeats ourselves
            } else {
                // Not a repeat.  Save key down state in case we do see a repeat later.
                resetKeyRepeatLocked();
                mKeyRepeatState.nextRepeatTime = entry->eventTime + mConfig.keyRepeatTimeout;
            }
            mKeyRepeatState.lastKeyEntry = entry;
            entry->refCount += 1;
        } else if (! entry->syntheticRepeat) {
            resetKeyRepeatLocked();
        }

        if (entry->repeatCount == 1) {
            entry->flags |= AKEY_EVENT_FLAG_LONG_PRESS;
        } else {
            entry->flags &= ~AKEY_EVENT_FLAG_LONG_PRESS;
        }

        entry->dispatchInProgress = true;

        logOutboundKeyDetailsLocked("dispatchKey - ", entry);
    }

    // Handle case where the policy asked us to try again later last time.
    if (entry->interceptKeyResult == KeyEntry::INTERCEPT_KEY_RESULT_TRY_AGAIN_LATER) {
        if (currentTime < entry->interceptKeyWakeupTime) {
            if (entry->interceptKeyWakeupTime < *nextWakeupTime) {
                *nextWakeupTime = entry->interceptKeyWakeupTime;
            }
            return false; // wait until next wakeup
        }
        entry->interceptKeyResult = KeyEntry::INTERCEPT_KEY_RESULT_UNKNOWN;
        entry->interceptKeyWakeupTime = 0;
    }

    // Give the policy a chance to intercept the key.
    if (entry->interceptKeyResult == KeyEntry::INTERCEPT_KEY_RESULT_UNKNOWN) {
        if (entry->policyFlags & POLICY_FLAG_PASS_TO_USER) {
            CommandEntry* commandEntry = postCommandLocked(
                    & InputDispatcher::doInterceptKeyBeforeDispatchingLockedInterruptible);
            if (mFocusedWindowHandle != NULL) {
                commandEntry->inputWindowHandle = mFocusedWindowHandle;
            }
            commandEntry->keyEntry = entry;
            entry->refCount += 1;
            return false; // wait for the command to run
        } else {
            entry->interceptKeyResult = KeyEntry::INTERCEPT_KEY_RESULT_CONTINUE;
        }
    } else if (entry->interceptKeyResult == KeyEntry::INTERCEPT_KEY_RESULT_SKIP) {
        if (*dropReason == DROP_REASON_NOT_DROPPED) {
            *dropReason = DROP_REASON_POLICY;
        }
    }

    // Clean up if dropping the event.
    if (*dropReason != DROP_REASON_NOT_DROPPED) {
        setInjectionResultLocked(entry, *dropReason == DROP_REASON_POLICY
                ? INPUT_EVENT_INJECTION_SUCCEEDED : INPUT_EVENT_INJECTION_FAILED);
        return true;
    }
```

```
    Vector<InputTarget> inputTargets;
    int32_t injectionResult = findFocusedWindowTargetsLocked(currentTime,
            entry, inputTargets, nextWakeupTime);
    if (injectionResult == INPUT_EVENT_INJECTION_PENDING) {
        return false;
    }

    setInjectionResultLocked(entry, injectionResult);
    if (injectionResult != INPUT_EVENT_INJECTION_SUCCEEDED) {
        return true;
    }

    addMonitoringTargetsLocked(inputTargets);

    // Dispatch the key.
    dispatchEventLocked(currentTime, entry, inputTargets);
    return true;
}
```

dispatchEventLocked会遍历inputTargets中所有InputTarget，每个InputTarget都对应着一个InputChannel，每个InputChannel对应着一个Connection，代表着InputDispatcher和Window的联系纽带。

/frameworks/native/services/inputflinger/InputDispatcher.cpp

```
void InputDispatcher::dispatchEventLocked(nsecs_t currentTime,
        EventEntry* eventEntry, const Vector<InputTarget>& inputTargets) {
#if DEBUG_DISPATCH_CYCLE
    ALOGD("dispatchEventToCurrentInputTargets");
#endif

    ALOG_ASSERT(eventEntry->dispatchInProgress); // should already have been set to true

    pokeUserActivityLocked(eventEntry);

    for (size_t i = 0; i < inputTargets.size(); i++) {
        const InputTarget& inputTarget = inputTargets.itemAt(i);

        ssize_t connectionIndex = getConnectionIndexLocked(inputTarget.inputChannel);
        if (connectionIndex >= 0) {
            sp<Connection> connection = mConnectionsByFd.valueAt(connectionIndex);
            prepareDispatchCycleLocked(currentTime, connection, eventEntry, &inputTarget);
        } else {
#if DEBUG_FOCUS
            ALOGD("Dropping event delivery to target with channel '%s' because it "
                    "is no longer registered with the input dispatcher.",
                    inputTarget.inputChannel->getName().string());
#endif
        }
    }
}
```

直接看enqueueDispatchEntriesLocked。

/frameworks/native/services/inputflinger/InputDispatcher.cpp

```
void InputDispatcher::prepareDispatchCycleLocked(nsecs_t currentTime,
        const sp<Connection>& connection, EventEntry* eventEntry, const InputTarget* inputTarget) {
#if DEBUG_DISPATCH_CYCLE
    ALOGD("channel '%s' ~ prepareDispatchCycle - flags=0x%08x, "
            "xOffset=%f, yOffset=%f, scaleFactor=%f, "
            "pointerIds=0x%x",
            connection->getInputChannelName(), inputTarget->flags,
            inputTarget->xOffset, inputTarget->yOffset,
            inputTarget->scaleFactor, inputTarget->pointerIds.value);
#endif

    // Skip this event if the connection status is not normal.
    // We don't want to enqueue additional outbound events if the connection is broken.
    if (connection->status != Connection::STATUS_NORMAL) {
#if DEBUG_DISPATCH_CYCLE
        ALOGD("channel '%s' ~ Dropping event because the channel status is %s",
                connection->getInputChannelName(), connection->getStatusLabel());
#endif
        return;
```

```
    if (inputTarget->flags & InputTarget::FLAG_SPLIT) {
        ALOG_ASSERT(eventEntry->type == EventEntry::TYPE_MOTION);

        MotionEntry* originalMotionEntry = static_cast<MotionEntry*>(eventEntry);
        if (inputTarget->pointerIds.count() != originalMotionEntry->pointerCount) {
            MotionEntry* splitMotionEntry = splitMotionEvent(
                    originalMotionEntry, inputTarget->pointerIds);
            if (!splitMotionEntry) {
                return; // split event was dropped
            }
#if DEBUG_FOCUS
            ALOGD("channel '%s' ~ Split motion event.",
                    connection->getInputChannelName());
            logOutboundMotionDetailsLocked(" ", splitMotionEntry);
#endif
            enqueueDispatchEntriesLocked(currentTime, connection,
                    splitMotionEntry, inputTarget);
            splitMotionEntry->release();
            return;
        }
    }

    // Not splitting.  Enqueue dispatch entries for the event as is.
    enqueueDispatchEntriesLocked(currentTime, connection, eventEntry, inputTarget);
}
```

enqueueDispatchEntriesLocked将构建一个DispatchEntry，push到Connection的outboundQueue队列中。如果Connection对应的outboundQueue之前为空，现在不为空的话，调用startDispatchCycleLocked进入推送过程。

/frameworks/native/services/inputflinger/InputDispatcher.cpp

```
void InputDispatcher::enqueueDispatchEntriesLocked(nsecs_t currentTime,
        const sp<Connection>& connection, EventEntry* eventEntry, const InputTarget* inputTarget) {
    bool wasEmpty = connection->outboundQueue.isEmpty();

    // Enqueue dispatch entries for the requested modes.
    enqueueDispatchEntryLocked(connection, eventEntry, inputTarget,
            InputTarget::FLAG_DISPATCH_AS_HOVER_EXIT);
    enqueueDispatchEntryLocked(connection, eventEntry, inputTarget,
            InputTarget::FLAG_DISPATCH_AS_OUTSIDE);
    enqueueDispatchEntryLocked(connection, eventEntry, inputTarget,
            InputTarget::FLAG_DISPATCH_AS_HOVER_ENTER);
    enqueueDispatchEntryLocked(connection, eventEntry, inputTarget,
            InputTarget::FLAG_DISPATCH_AS_IS);
    enqueueDispatchEntryLocked(connection, eventEntry, inputTarget,
            InputTarget::FLAG_DISPATCH_AS_SLIPPERY_EXIT);
    enqueueDispatchEntryLocked(connection, eventEntry, inputTarget,
            InputTarget::FLAG_DISPATCH_AS_SLIPPERY_ENTER);

    // If the outbound queue was previously empty, start the dispatch cycle going.
    if (wasEmpty && !connection->outboundQueue.isEmpty()) {
        startDispatchCycleLocked(currentTime, connection);
    }
}
```

/frameworks/native/services/inputflinger/InputDispatcher.cpp

```
void InputDispatcher::enqueueDispatchEntryLocked(
        const sp<Connection>& connection, EventEntry* eventEntry, const InputTarget* inputTarget,
        int32_t dispatchMode) {
    int32_t inputTargetFlags = inputTarget->flags;
    if (!(inputTargetFlags & dispatchMode)) {
        return;
    }
    inputTargetFlags = (inputTargetFlags & ~InputTarget::FLAG_DISPATCH_MASK) | dispatchMode;

    // This is a new event.
    // Enqueue a new dispatch entry onto the outbound queue for this connection.
    DispatchEntry* dispatchEntry = new DispatchEntry(eventEntry, // increments ref
            inputTargetFlags, inputTarget->xOffset, inputTarget->yOffset,
            inputTarget->scaleFactor);

    // Apply target flags and update the connection's input state.
    switch (eventEntry->type) {
```

```
        dispatchEntry->resolvedFlags = keyEntry->flags;

        if (!connection->inputState.trackKey(keyEntry,
                dispatchEntry->resolvedAction, dispatchEntry->resolvedFlags)) {
#if DEBUG_DISPATCH_CYCLE
            ALOGD("channel '%s' ~ enqueueDispatchEntryLocked: skipping inconsistent key event",
                    connection->getInputChannelName());
#endif
            delete dispatchEntry;
            return; // skip the inconsistent event
        }
        break;
    }

    case EventEntry::TYPE_MOTION: {
        MotionEntry* motionEntry = static_cast<MotionEntry*>(eventEntry);
        if (dispatchMode & InputTarget::FLAG_DISPATCH_AS_OUTSIDE) {
            dispatchEntry->resolvedAction = AMOTION_EVENT_ACTION_OUTSIDE;
        } else if (dispatchMode & InputTarget::FLAG_DISPATCH_AS_HOVER_EXIT) {
            dispatchEntry->resolvedAction = AMOTION_EVENT_ACTION_HOVER_EXIT;
        } else if (dispatchMode & InputTarget::FLAG_DISPATCH_AS_HOVER_ENTER) {
            dispatchEntry->resolvedAction = AMOTION_EVENT_ACTION_HOVER_ENTER;
        } else if (dispatchMode & InputTarget::FLAG_DISPATCH_AS_SLIPPERY_EXIT) {
            dispatchEntry->resolvedAction = AMOTION_EVENT_ACTION_CANCEL;
        } else if (dispatchMode & InputTarget::FLAG_DISPATCH_AS_SLIPPERY_ENTER) {
            dispatchEntry->resolvedAction = AMOTION_EVENT_ACTION_DOWN;
        } else {
            dispatchEntry->resolvedAction = motionEntry->action;
        }
        if (dispatchEntry->resolvedAction == AMOTION_EVENT_ACTION_HOVER_MOVE
                && !connection->inputState.isHovering(
                        motionEntry->deviceId, motionEntry->source, motionEntry->displayId)) {
#if DEBUG_DISPATCH_CYCLE
        ALOGD("channel '%s' ~ enqueueDispatchEntryLocked: filling in missing hover enter event",
                connection->getInputChannelName());
#endif
            dispatchEntry->resolvedAction = AMOTION_EVENT_ACTION_HOVER_ENTER;
        }

        dispatchEntry->resolvedFlags = motionEntry->flags;
        if (dispatchEntry->targetFlags & InputTarget::FLAG_WINDOW_IS_OBSCURED) {
            dispatchEntry->resolvedFlags |= AMOTION_EVENT_FLAG_WINDOW_IS_OBSCURED;
        }

        if (!connection->inputState.trackMotion(motionEntry,
                dispatchEntry->resolvedAction, dispatchEntry->resolvedFlags)) {
#if DEBUG_DISPATCH_CYCLE
            ALOGD("channel '%s' ~ enqueueDispatchEntryLocked: skipping inconsistent motion event",
                    connection->getInputChannelName());
#endif
            delete dispatchEntry;
            return; // skip the inconsistent event
        }
        break;
    }
    }

    // Remember that we are waiting for this dispatch to complete.
    if (dispatchEntry->hasForegroundTarget()) {
        incrementPendingForegroundDispatchesLocked(eventEntry);
    }

    // Enqueue the dispatch entry.
    connection->outboundQueue.enqueueAtTail(dispatchEntry);
    traceOutboundQueueLengthLocked(connection);
```

startDispatchCycleLocked会取出outboundQueue中的每一个对应的DispatchEntry，使用Connection的InputPublisher的publishKeyEvent函数将这个DispatchEntry携带的信息组合成一个InputMessage，推送到InputChannel中。

/frameworks/native/services/inputflinger/InputDispatcher.cpp

```
void InputDispatcher::startDispatchCycleLocked(nsecs_t currentTime,
        const sp<Connection>& connection) {
#if DEBUG_DISPATCH_CYCLE
    ALOGD("channel '%s' ~ startDispatchCycle"
```

```
    while (connection->status == Connection::STATUS_NORMAL
            && !connection->outboundQueue.isEmpty()) {
        DispatchEntry* dispatchEntry = connection->outboundQueue.head;
        dispatchEntry->deliveryTime = currentTime;

        // Publish the event.
        status_t status;
        EventEntry* eventEntry = dispatchEntry->eventEntry;
        switch (eventEntry->type) {
        case EventEntry::TYPE_KEY: {
            KeyEntry* keyEntry = static_cast<KeyEntry*>(eventEntry);

            // Publish the key event.
            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);
            break;
        }

        case EventEntry::TYPE_MOTION: {
            MotionEntry* motionEntry = static_cast<MotionEntry*>(eventEntry);

            PointerCoords scaledCoords[MAX_POINTERS];
            const PointerCoords* usingCoords = motionEntry->pointerCoords;

            // Set the X and Y offset depending on the input source.
            float xOffset, yOffset, scaleFactor;
            if ((motionEntry->source & AINPUT_SOURCE_CLASS_POINTER)
                    && !(dispatchEntry->targetFlags & InputTarget::FLAG_ZERO_COORDS)) {
                scaleFactor = dispatchEntry->scaleFactor;
                xOffset = dispatchEntry->xOffset * scaleFactor;
                yOffset = dispatchEntry->yOffset * scaleFactor;
                if (scaleFactor != 1.0f) {
                    for (uint32_t i = 0; i < motionEntry->pointerCount; i++) {
                        scaledCoords[i] = motionEntry->pointerCoords[i];
                        scaledCoords[i].scale(scaleFactor);
                    }
                    usingCoords = scaledCoords;
                }
            } else {
                xOffset = 0.0f;
                yOffset = 0.0f;
                scaleFactor = 1.0f;

                // We don't want the dispatch target to know.
                if (dispatchEntry->targetFlags & InputTarget::FLAG_ZERO_COORDS) {
                    for (uint32_t i = 0; i < motionEntry->pointerCount; i++) {
                        scaledCoords[i].clear();
                    }
                    usingCoords = scaledCoords;
                }
            }

            // Publish the motion event.
            status = connection->inputPublisher.publishMotionEvent(dispatchEntry->seq,
                    motionEntry->deviceId, motionEntry->source,
                    dispatchEntry->resolvedAction, motionEntry->actionButton,
                    dispatchEntry->resolvedFlags, motionEntry->edgeFlags,
                    motionEntry->metaState, motionEntry->buttonState,
                    xOffset, yOffset, motionEntry->xPrecision, motionEntry->yPrecision,
                    motionEntry->downTime, motionEntry->eventTime,
                    motionEntry->pointerCount, motionEntry->pointerProperties,
                    usingCoords);
            break;
        }

        default:
            ALOG_ASSERT(false);
            return;
        }

        // Check the result.
```

```
                    ALOGE("channel '%s' ~ Could not publish event because the pipe is full. "
                            "This is unexpected because the wait queue is empty, so the pipe "
                            "should be empty and we shouldn't have any problems writing an "
                            "event to it, status=%d", connection->getInputChannelName(), status);
                    abortBrokenDispatchCycleLocked(currentTime, connection, true /*notify*/);
                } else {
                    // Pipe is full and we are waiting for the app to finish process some events
                    // before sending more events to it.
#if DEBUG_DISPATCH_CYCLE
                    ALOGD("channel '%s' ~ Could not publish event because the pipe is full, "
                            "waiting for the application to catch up",
                            connection->getInputChannelName());
#endif
                    connection->inputPublisherBlocked = true;
                }
            } else {
                ALOGE("channel '%s' ~ Could not publish event due to an unexpected error, "
                        "status=%d", connection->getInputChannelName(), status);
                abortBrokenDispatchCycleLocked(currentTime, connection, true /*notify*/);
            }
            return;
        }

        // Re-enqueue the event on the wait queue.
        connection->outboundQueue.dequeue(dispatchEntry);
        traceOutboundQueueLengthLocked(connection);
        connection->waitQueue.enqueueAtTail(dispatchEntry);
        traceWaitQueueLengthLocked(connection);
    }
}
```

在窗口的创建过程中，会通过openInputChannelPair创建一对UNIX域套接字。client端即窗口端保存的是inputChannels[0]，server端即应用程序端View保存的inputChannels[1]。这些java层的InputChannel在native有着对应的InputChannel。

registerInputChannel的作用是注册InputChanel。在往下调注册InputChannel的时候，会new一个Connection,标示一对连接，并且会将对应的fd和Connection作为键值对保存到mConnectionsByFd中。Connection的构造函数会new一个inputPublisher。所以Connection含有输入窗口句柄InputWindowHandle，连接纽带InputChannel，推送工具InputPublisher，Key保存队列Queue< DispatchEntry>和Key等待队列Queue< DispatchEntry>等成员。

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

```
    public int addWindow(Session session, IWindow client, int seq,
            WindowManager.LayoutParams attrs, int viewVisibility, int displayId,
            Rect outContentInsets, Rect outStableInsets, Rect outOutsets,
            InputChannel outInputChannel) {
            ...
            if (outInputChannel != null && (attrs.inputFeatures
                    & WindowManager.LayoutParams.INPUT_FEATURE_NO_INPUT_CHANNEL) == 0) {
                String name = win.makeInputChannelName();
                InputChannel[] inputChannels = InputChannel.openInputChannelPair(name);
                win.setInputChannel(inputChannels[0]);
                inputChannels[1].transferTo(outInputChannel);

                mInputManager.registerInputChannel(win.mInputChannel, win.mInputWindowHandle);
            }
```

/frameworks/native/services/inputflinger/InputDispatcher.cpp

```
status_t InputDispatcher::registerInputChannel(const sp<InputChannel>& inputChannel,
        const sp<InputWindowHandle>& inputWindowHandle, bool monitor) {
#if DEBUG_REGISTRATION
    ALOGD("channel '%s' ~ registerInputChannel - monitor=%s", inputChannel->getName().string(),
            toString(monitor));
#endif

    { // acquire lock
        AutoMutex _l(mLock);

        if (getConnectionIndexLocked(inputChannel) >= 0) {
            ALOGW("Attempted to register already registered input channel '%s'",
                    inputChannel->getName().string());
            return BAD_VALUE;
        }

        sp<Connection> connection = new Connection(inputChannel, inputWindowHandle, monitor);
```

```

        if (monitor) {
            mMonitoringChannels.push(inputChannel);
        }

        mLooper->addFd(fd, 0, ALOOPER_EVENT_INPUT, handleReceiveCallback, this);
    } // release lock

    // Wake the looper because some connections have changed.
    mLooper->wake();
    return OK;
}
```

 ；现在已经了解了Connection的内部构造。看看Connection内部InputPublisher::publishKeyEvent实现。

/frameworks/native/services/inputflinger/InputDispatcher.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) {
#if DEBUG_TRANSPORT_ACTIONS
    ALOGD("channel '%s' publisher ~ publishKeyEvent: seq=%u, deviceId=%d, source=0x%x, "
            "action=0x%x, flags=0x%x, keyCode=%d, scanCode=%d, metaState=0x%x, repeatCount=%d,"
            "downTime=%lld, eventTime=%lld",
            mChannel->getName().string(), seq,
            deviceId, source, action, flags, keyCode, scanCode, metaState, repeatCount,
            downTime, eventTime);
#endif

    if (!seq) {
        ALOGE("Attempted to publish a key event with sequence number 0.");
        return BAD_VALUE;
    }

    InputMessage msg;
    msg.header.type = InputMessage::TYPE_KEY;
    msg.body.key.seq = seq;
    msg.body.key.deviceId = deviceId;
    msg.body.key.source = source;
    msg.body.key.action = action;
    msg.body.key.flags = flags;
    msg.body.key.keyCode = keyCode;
    msg.body.key.scanCode = scanCode;
    msg.body.key.metaState = metaState;
    msg.body.key.repeatCount = repeatCount;
    msg.body.key.downTime = downTime;
    msg.body.key.eventTime = eventTime;
    return mChannel->sendMessage(&msg);
}
```

InputChannel::sendMessage就是将封装好的InputMessage通过send推送到mFd中去。那么，在server端的InputChannel会接收到这个InputMessage。

/frameworks/native/services/inputflinger/InputDispatcher.cpp

```
status_t InputChannel::sendMessage(const InputMessage* msg) {
    size_t msgLength = msg->size();
    ssize_t nWrite;
    do {
        nWrite = ::send(mFd, msg, msgLength, MSG_DONTWAIT | MSG_NOSIGNAL);
    } while (nWrite == -1 && errno == EINTR);

    if (nWrite < 0) {
        int error = errno;
#if DEBUG_CHANNEL_MESSAGES
        ALOGD("channel '%s' ~ error sending message of type %d, errno=%d", mName.string(),
                msg->header.type, error);
```

```
            return WOULD_BLOCK;
        }
        if (error == EPIPE || error == ENOTCONN || error == ECONNREFUSED || error == ECONNRESET) {
            return DEAD_OBJECT;
        }
        return -error;
    }

    if (size_t(nWrite) != msgLength) {
#if DEBUG_CHANNEL_MESSAGES
        ALOGD("channel '%s' ~ error sending message type %d, send was incomplete",
                mName.string(), msg->header.type);
#endif
        return DEAD_OBJECT;
    }

#if DEBUG_CHANNEL_MESSAGES
    ALOGD("channel '%s' ~ sent message of type %d", mName.string(), msg->header.type);
#endif
    return OK;
}
```

至此，派发流程从InputDispatcher的派发中走到了View的逻辑中。

Android 5.0输入系统分析之InputDispatcher线程分析 阅读数 252

上编分析知道是InputReader线程唤醒了InputDispatcher线程分析，InputDispatcher线程是从dispatchOnce启动... 博文 来自： 100度多0点01度

Android按键超时的ANR原理小结 阅读数 6905

1) 按键处理主要的功臣是WindowManagerService.mInputManager@WindowManagerService.javamInputMan... 博文 来自： wuhengde的专栏

android Input系统事件分发流程 阅读数 543

最近被问到androidInput系统事件分发流程，虽然有看过相关源码，但是只是匆匆一瞥，没什么印象，解释了半天自... 博文 来自： 星辰旋风的博客

android 事件派发流程详解 阅读数 713

Android5.0(Lollipop)事件输入系统(InputSystem)2014-12-15 23 个评论 来源：世事难料，保持低调 收藏 我... 博文 来自： feitian_666的博客

InputManagerService之事件的初始化与分发 阅读数 1028

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

Android系统触屏事件传递派发浅析（二） 阅读数 1543

上一篇文章提到在InputDispatcher中，connection调用inputPublisher.publishMotionEvent后分发就完成了，然... 博文 来自： karizhang的专栏

Android系统--事件读取及分发 阅读数 2441

1.简介 WindowManagerService分发事件是通过它的InputManager来完成的。 在初始化时，各部分状态如下... 博文 来自： sadamoo的专栏

InputDispatcher异常，导致程序闪退，调试一个星期了，求高手 04-29

症状：操作的时候，拉一个列表拉着拉着，程序就崩溃了（不固定是拉动列表），没错误提示，直接返回桌面。注册了全局... 论坛

Android输入事件InputReader和InputDispatcher分析 阅读数 392

.C++frameworks/native/services/inputflinger/InputDispatcher.cppvoidInputDispatcher::notifyKey(constN... 博文 来自： unbroken

InputDispatcher： MainActivity (server)' ~ Channel is unrecoverably broken and will b... 阅读数 7393

log日志如下：10-2722:03:38.1903338-4531/?I/HwPowerMonitor.PowerExceptionObservable：unhandledon... 博文 来自： yangyao_iphone...

Android Input系统的启动以及Input场景下的ANR 阅读数 1289

为了分析Input场景下ANR发生的原因，特意找了对Input系统全面介绍的一篇文章，如果系统对于InputEvent超过... 博文 来自： 专栏

Android ANR 问题第二弹------Input事件是如何超时导致ANR的 阅读数 2187

在AndroidANR问题第一弹中，我们介绍AndroidANR问题分为三类：Input，Receiver，Service。我们现在就先来... 博文 来自： 为码消得人憔悴

Android输入输出系统之TouchEvent流程 阅读数 18

记不住密码怎么办？http://a.app.qq.com/o/simple.jsp?pkgname=com.wa505.kf.epassword 一个是InputRead... 博文 来自： daojin505的专栏

从input tap来看事件注入的调用流程 阅读数 2038

在应用中进行inputtap300600这样的调用，不能在其他应用的窗口上进行事件注入，这里我们通过查看log来了解其... 博文 来自： aaajj的专栏

Android Input流程分析（三）：InputReader 阅读数 764

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

ANR源码分析之InputDispatcher Timeout 阅读数 662

在上篇文章中，介绍ANR产生的原因、ANR的分类以及ANR问题的分析。本篇文章接下来将从源码的角度来分析AN... 博文 来自： chewbee的专栏

Android输入事件从读取到分发四：InputDispatcherThread发送事件到View结构树的过程 阅读数 1819

通过上一节的分析，我们发现InputDispatcherThread使用InputChannel的sendMessage方法发送了一条消息，但... 博文 来自： 阳光玻璃杯

Android4.4——InputManagerService之InputDispatcher线程 阅读数 1062

上篇博客《Android4.4——InputManagerService之InputReader线程》的最后已经进入到InputDispatcher线程中... 博文 来自： WellYiu的专栏

Android 事件捕捉和处理流程分析 阅读数 1万+

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

Android入门之把窗口信息传递给InputDispatcher 阅读数 2312

当窗口信息变化时，就应该把变化后的窗口信息传递给InputDispatcher，所以，从“什么时候会改变窗口信息入手... 博文 来自： Roger Luo

模拟点击的方法实现视频监控功能（完整版） 阅读数 2544

对房屋等进行视频监控有较大的需求，现在手机较多，怎么样用手机去作为监控器实现这个功能呢？比较便捷的一种... 博文 来自： aaajj的专栏

Android OTA升级原理和流程分析（四）---Android系统Recovery模式的工作原理 阅读数 5484

AndroidOTA升级原理和流程分析（四）---Android系统Recovery模式的工作原理 在使用update.zip包升级时怎... 博文 来自： ylyuanlu的专栏

android ANR源码分析 --- 之三 阅读数 1400

4,inputDispatchingTimeout当input事件处理得慢就会触发ANR.ANR时间区别便是指当前这次的事件dispatch过程... 博文 来自： Jack的博客

Android之Input子系统按键repeat 阅读数 5848

Android系统中长按键部分：Linux驱动只是在起初按下时上报个down事件，在抬起后再报个up事件；其中，不会在... 博文 来自： 程序改变生活

Android6.0 按键流程（三）InputDispatcher分发输入消息 阅读数 3467

上一篇博客分析了InputReader中扫描码与键盘码的转化，今天我们再来分析下InputDispatcher 一、InputDispatc... 博文 来自： kc58236582的博客

Android中InputManagerService里的InputReader和inputDispatcher 阅读数 3344

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

大伙帮忙看看 内存溢出导致android重启。 01-10

因为Out of memory on a 124216-byte allocation. 没有足够内存分配，导致系统服务挂掉，引起android系统重启，各... 论坛

E/InputDispatcher: channel '32d434a4 Toast (server)' ~ Channel is unrecoverably brok... 阅读数 2079

报错：E/InputDispatcher:channel ‘32d434a4Toast(server)’ ~Channelisunrecoverablybrokenandwillbedispo... 博文 来自： willba的博客

android 关于InputDispatcher出现Consumer异常的解决方法 阅读数 19

10-2303:24:46.346:ERROR/InputDispatcher(61):channel'40774ac8com.marsor.meinv.panfa/com.marsor.mei... 博文 来自： 我思故我在

Android 4.1 Input设备流程分析 03-25

Android 4.1 Input设备流程分析包含触摸流程的详细分析图 下载

input 按键分发 阅读数 1456

文章出处：http://blog.csdn.net/shift_wwx请转载的朋友标明出处~~之前InputManager的启动过程已经对inputM... 博文 来自： 私房菜之 --学--无-...

安卓框架，分析解决项目中出现的anr 阅读数 655

先看下anr日志07-1615:31:47.551E/ActivityManager(1775):Reason:Inputdispatchingtimedout(Waitingbecaus... 博文 来自： u013762045的博客

jni 内存溢出 阅读数 194

ndk内存溢出原因:jbytejbarray=(jbyte)malloc(len*sizeof(jbyte));没有及时释放内存log日志现象:SystemDeviceSer... 博文 来自： qq_34705828的博客

android 系统触摸屏BUG解决过程分析 阅读数 2686

BUG描述:添加触摸屏驱动后,apk对触摸事件没有响应. Linux层驱动移植内核根目录makemenuconfig更改“Device... 博文 来自： welljrj的博客

Android键值上报流程 阅读数 6712

一、介绍 在常用手机中，常用的键值有power,volume_up,volume_down，home，back，menu。其中power先... 博文 来自： 羽凌寒

andorid 问题集合 阅读数 947

1.有些设备出现下面文字：Droppingeventbecausethereisnotouchablewindowat(908,546).把img_qp.setOnClic... 博文 来自： laose307的专栏

基于PyTorch的深度学习入门教程（六）——数据并行化 阅读数 4720

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

Android实现QQ分享及注意事项 阅读数 5547

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

CNN笔记：通俗理解卷积神经网络 阅读数 21万+

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

Android开发本地及网络Mp3音乐播放器(十二)创建NetMusicListAdapter、SearchResult显示... 阅读数 9291

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

【小程序】微信小程序开发实践 阅读数 25万+

HttpClient使用详解 阅读数 80万+

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

0

android客户端与服务器端交互 如何保持session 阅读数 4万+

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

三菱FX系列PLC与PC通讯的实现之专有协议（计算机联接）的程序设计之一 阅读数 1万+

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

如何在ArcGIS Online中构建自己的应用程序模板初级篇-显示地图 阅读数 4万+

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

再谈iOS 7的手势滑动返回功能 阅读数 8万+

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

[ASP.NET]二维码的创建 阅读数 5439

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

jquery/js实现一个网页同时调用多个倒计时(最新的) 阅读数 44万+

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

将Excel文件导入数据库（POI+Excel+MySQL+jsp页面导入）第一次优化 阅读数 3万+

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

webService学习（二）—— 调用自定义对象参数 阅读数 2万+

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

人脸检测工具face_recognition的安装与应用 阅读数 5万+

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

C#实现开发windows服务实现自动从FTP服务器下载文件（自行设置分/时执行） 阅读数 2万+

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

eclipse复制粘贴卡死 阅读数 2488

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

微信支付V3微信公众号支付PHP教程(thinkPHP5公众号支付)/JSSDK的使用 阅读数 14万+

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

linux上安装Docker(非常简单的安装方法) 阅读数 20万+

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

如何在ubuntu 16.04上安装 RealSense（相机型号：Intel SR300） 阅读数 3633

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

openfire 3.8.2 源码部署 /开发配置 / 二次开发 阅读数 6643

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... 博文 来自： StillCity的专栏

强连通分量及缩点tarjan算法解析 阅读数 57万+

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

【HTTP】Fiddler（一）- Fiddler简介 阅读数 30万+

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

centos 查看命令源码 阅读数 8万+

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

OpenCV+OpenGL 双目立体视觉三维重建 阅读数 4万+

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

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

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

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

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

Hadoop+HBase完全分布式安装 阅读数 4313

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

DataTables 的 实例 《一》 阅读数 1万+

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

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

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

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

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

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

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

JavaWeb多文件上传及zip打包下载 阅读数 6536

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

R语言逻辑回归、ROC曲线和十折交叉验证 阅读数 5万+

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

0

Invoker123 关注

原创	粉丝	喜欢	评论
26	14	5	0

等级： 博客 2 访问： 1万+

积分： 410 排名： 15万+

勋章：

最新文章

个人分类

归档

Android Sensor HAL层分析
阅读数 2218

SEAndroid流程分析
阅读数 1176

process_config解析fstab文件
阅读数 800

Android Input流程分析（三）：InputReader
阅读数 757

SurfaceFlinger原理（一）：SurfaceFlinger的初始化
阅读数 713

0