简述

- 跟进短信接收流程和jira的初步使用

接收流程

1. 接收到UNSOL_RESPONSE_NEW_SMS消息
    - hardware/ril/libril/ril_unsol_commands.h定义了接收的消息类型对应的处理方法
    - 具体的处理在hardware/ril/libril/ril_service.cpp中，通过IRadioIndication.newSms()处理
2. IRadioIndication.newSms()跨进程调用
    - IRadioIndication.hidl位于hardware/interfaces/radio/1.0/IRadioIndication.hal
    - IRadioIndication.hidl的实现在 frameworks/opt/telephony/src/java/com/android/internal/telephony/RadioIndication.java
    - newSms():Indicates when new SMS is received.
    - ```
      public void newSms(int indicationType, ArrayList<Byte> pdu) {
          mRil.processIndication(indicationType);
          //将ArrayList<Byte>转换为byte[]
          byte[] pduArray = RIL.arrayListToPrimitiveArray(pdu);
          if (RIL.RILJ_LOGD) mRil.unsljLog(RIL_UNSOL_RESPONSE_NEW_SMS);

          SmsMessageBase smsb =
      com.android.internal.telephony.gsm.SmsMessage.createFromPdu(pduArray);
          if (mRil.mGsmSmsRegistrant != null) {
              //protected Registrant mGsmSmsRegistrant;
              mRil.mGsmSmsRegistrant.notifyRegistrant(
                      new AsyncResult(null, smsb == null ? null : new
      SmsMessage(smsb), null));
          }
      }

      //mGsmSmsRegistrant的注册
          //在GsmInboundSmsHandler的构造方法中调用了ril的setOnNewGSmSms()
          protected GsmInboundSmsHandler(Context context, SmsStorageMonitor
      storageMonitor,
                  Phone phone) {
              super("GsmInboundSmsHandler", context, storageMonitor, phone);
          // MTK-END
              phone.mCi.setOnNewGsmSms(getHandler(), EVENT_NEW_SMS, null);
              mDataDownloadHandler = new UsimDataDownloadHandler(phone.mCi,
      phone.getPhoneId());
              mCellBroadcastServiceManager.enable();

              if (TEST_MODE) {
                  if (sTestBroadcastReceiver == null) {
                      sTestBroadcastReceiver = new
      GsmCbTestBroadcastReceiver();
                      IntentFilter filter = new IntentFilter();
                      filter.addAction(TEST_ACTION);
      ```
      ```

```
                context.registerReceiver(sTestBroadcastReceiver,
filter);
            }
        }
    }

    //ril是CommandsInterface，BaseCommands实现了CommandsInterface接口
    //BaseCommands.setOnNewGsmSms（）
    public void setOnNewGsmSms(Handler h, int what, Object obj) {
        mGsmSmsRegistrant = new Registrant (h, what, obj);
    }
```

3. Registrant的notifyRegistrant()
   - 位于frameworks/base/core/java/android/os/Registrant.java
   - ```
     public void notifyRegistrant(AsyncResult ar)
     {
         internalNotifyRegistrant (ar.result, ar.exception);
     }

     void internalNotifyRegistrant (Object result, Throwable exception)
     {
         Handler h = getHandler();

         if (h == null) {
             clear();
         } else {
             Message msg = Message.obtain();

             msg.what = what;
             msg.obj = new AsyncResult(userObj, result, exception);
             //发出消息
             h.sendMessage(msg);
         }
     }
     ```

4. GsmInboundSmsHandler响应消息请求
   - GsmInboundSmsHandler继承自InboundSmsHandler。InboundSmsHandler继承自StateMachine。InboundSmsHandler有五个状态，分别为default、startup、idle、delivering、waiting。接收到短信，从一开始的idle状态进入delivering状态。
     - DeliveringState：In the delivering state, the inbound SMS is processed and stored in the raw table.The message is acknowledged before we exit this state. If there is a message to broadcast,transition to **WaitingState** state to send the ordered broadcast and wait for the results. When all messages have been processed, the halting state will release the wakelock.
   - GsmInboundSmsHandler位于frameworks/opt/telephony/src/java/com/android/internal/telephony/gsm/GsmInboundSmsHandler.java
   - ```
     //InboundSmsHandler的内部类DeliveringState的processMessage()响应EVENT_NEW_SMS消息
     public boolean processMessage(Message msg) {
     ```

```
    if (DBG) log("DeliveringState.processMessage: processing " +
getWhatToString(msg.what));
    switch (msg.what) {
        case EVENT_NEW_SMS:
            // handle new SMS from RIL
            handleNewSms((AsyncResult) msg.obj);
            sendMessage(EVENT_RETURN_TO_IDLE);
            return HANDLED;
    }
}

private void handleNewSms(AsyncResult ar) {
    if (ar.exception != null) {
        loge("Exception processing incoming SMS: " + ar.exception);
        return;
    }

    int result;
    try {
        SmsMessage sms = (SmsMessage) ar.result;
        result = dispatchMessage(sms.mWrappedSmsMessage,
SOURCE_NOT_INJECTED);
    } catch (RuntimeException ex) {
        loge("Exception dispatching message", ex);
        result = RESULT_SMS_DISPATCH_FAILURE;
    }

    // RESULT_OK means that the SMS will be acknowledged by special
handling,
    // e.g. for SMS-PP data download. Any other result, we should ack
here.
    if (result != Activity.RESULT_OK) {
        boolean handled = (result == Intents.RESULT_SMS_HANDLED);
        notifyAndAcknowledgeLastIncomingSms(handled, result, null);
    }
}

private int dispatchMessage(SmsMessageBase smsb, @SmsSource int
smsSource) {
    int result = dispatchMessageRadioSpecific(smsb, smsSource);

    // In case of error, add to metrics. This is not required in case of
success, as the
    // data will be tracked when the message is processed
(processMessagePart).
    if (result != Intents.RESULT_SMS_HANDLED && result !=
Activity.RESULT_OK) {
        mMetrics.writeIncomingSmsError(mPhone.getPhoneId(), is3gpp2(),
smsSource, result);
        mPhone.getSmsStats().onIncomingSmsError(is3gpp2(), smsSource,
result);
    }
    return result;
}
```

5. InboundSmsHandler的子类GsmInboundSmsHandler的dispatchMessageRadioSpecific()方法

```
protected int dispatchMessageRadioSpecific(SmsMessageBase smsb,
@SmsSource int smsSource) {
    SmsMessage sms = (SmsMessage) smsb;
    return dispatchNormalMessage(smsb, smsSource);
}
```

6. InboundSmsHandler的内部类SmsBroadcastReceiver接收Intents.SMS_DELIVER_ACTION广播

```
//InboundSmsHandler的dispatchNormalMessage()方法
/*Dispatch a normal incoming SMS. This is called from
dispatchMessageRadioSpecific.
if no format-specific handling was required. Saves the PDU to the SMS
provider raw table,creates an InboundSmsTracker, then sends it to the
state machine as an EVENT_BROADCAST_SMS. Returns
Intents#RESULT_SMS_HANDLED or an error value.
*/
protected int dispatchNormalMessage(SmsMessageBase sms, @SmsSource int
smsSource) {
    SmsHeader smsHeader = sms.getUserDataHeader();
    /*InboundSmsTracker:Tracker for an incoming SMS message ready to
broadcast to listeners.This is similar to
com.android.internal.telephony.SMSDispatcher.SmsTracker used for
outgoing messages.*/
    InboundSmsTracker tracker;

    if ((smsHeader == null) || (smsHeader.concatRef == null)) {
        // Message is not concatenated.
        int destPort = -1;
        if (smsHeader != null && smsHeader.portAddrs != null) {
            // The message was sent to a port.
            destPort = smsHeader.portAddrs.destPort;
            if (DBG) log("destination port: " + destPort);
        }
        tracker = TelephonyComponentFactory.getInstance()
                .inject(InboundSmsTracker.class.getName())
                .makeInboundSmsTracker(mContext, sms.getPdu(),
                        sms.getTimestampMillis(), destPort, is3gpp2(),
false,
                        sms.getOriginatingAddress(),
sms.getDisplayOriginatingAddress(),
                        sms.getMessageBody(), sms.getMessageClass() ==
MessageClass.CLASS_0,
                        mPhone.getSubId(), smsSource);
    } else {
        // Create a tracker for this message segment.
        SmsHeader.ConcatRef concatRef = smsHeader.concatRef;
        SmsHeader.PortAddrs portAddrs = smsHeader.portAddrs;
        int destPort = (portAddrs != null ? portAddrs.destPort : -1);
        tracker = TelephonyComponentFactory.getInstance()
                .inject(InboundSmsTracker.class.getName())
                .makeInboundSmsTracker(mContext, sms.getPdu(),
                        sms.getTimestampMillis(), destPort, is3gpp2(),
```

```
                        sms.getOriginatingAddress(),
sms.getDisplayOriginatingAddress(),
                        concatRef.refNumber, concatRef.seqNumber,
concatRef.msgCount, false,
                        sms.getMessageBody(), sms.getMessageClass() ==
MessageClass.CLASS_0,
                        mPhone.getSubId(), smsSource);
    }

    if (VDBG) log("created tracker: " + tracker);

    // de-duping is done only for text messages
    // destPort = -1 indicates text messages, otherwise it's a data sms
    return
addTrackerToRawTableAndSendMessage(tracker,tracker.getDestPort() == -1
/* de-dup if text message */);
}

//Helper to add the tracker to the raw table and then send a message to
broadcast it, if successful. Returns the SMS intent status to return to
the SMSC.
protected int addTrackerToRawTableAndSendMessage(InboundSmsTracker
tracker, boolean deDup) {
    //记录数据库
    int result = addTrackerToRawTable(tracker, deDup);
    switch(result) {
        case Intents.RESULT_SMS_HANDLED:
            sendMessage(EVENT_BROADCAST_SMS, tracker);
            return Intents.RESULT_SMS_HANDLED;

        case Intents.RESULT_SMS_DUPLICATED:
            return Intents.RESULT_SMS_HANDLED;

        default:
            return result;
    }
}

//InboundSmsHandler的内部类DeliveringState的processMessage()响应
EVENT_BROADCAST_SMS消息
public boolean processMessage(Message msg) {
    if (DBG) log("DeliveringState.processMessage: processing " +
getWhatToString(msg.what));
    switch (msg.what) {
        case EVENT_BROADCAST_SMS:
            // if any broadcasts were sent, transition to waiting state
            InboundSmsTracker inboundSmsTracker = (InboundSmsTracker)
msg.obj;
            if (processMessagePart(inboundSmsTracker)) {
                sendMessage(obtainMessage(EVENT_UPDATE_TRACKER,
msg.obj));
                //mWaitingState主要是对新短信广播超时的处理
                transitionTo(mWaitingState);
            } else {
```

```
                // if event is sent from
SmsBroadcastUndelivered.broadcastSms(), and
                // processMessagePart() returns false, the state machine
will be stuck in
                // DeliveringState until next message is received. Send
message to
                // transition to idle to avoid that so that wakelock can
be released
                log("DeliveringState.processMessage:
EVENT_BROADCAST_SMS: No broadcast "
                        + "sent. Return to IdleState");
                sendMessage(EVENT_RETURN_TO_IDLE);
            }
            return HANDLED;
    }
}

/*Process the inbound SMS segment. If the message is complete, send it
as an ordered broadcast to interested receivers and return true. If the
message is a segment of an incomplete multi-part SMS, return false.*/
protected boolean processMessagePart(InboundSmsTracker tracker) {
// MTK-END
    int messageCount = tracker.getMessageCount();
    byte[][] pdus;
    long[] timestamps;
    int destPort = tracker.getDestPort();
    boolean block = false;
    String address = tracker.getAddress();
    //根据messageCount执行相对应的single或multipart处理逻辑
    ...
    SmsBroadcastReceiver resultReceiver =
tracker.getSmsBroadcastReceiver(this);

    // Always invoke SMS filters, even if the number ends up being
blocked, to prevent
    // surprising bugs due to blocking numbers that happen to be used
for visual voicemail SMS
    // or other carrier system messages.
    boolean filterInvoked = filterSms(
            pdus, destPort, tracker, resultReceiver, true /*
userUnlocked */, block);

    if (!filterInvoked) {
        // Block now if the filter wasn't invoked. Otherwise, it will be
the responsibility of
        // the filter to delete the SMS once processing completes.
        if (block) {
            deleteFromRawTable(tracker.getDeleteWhere(),
tracker.getDeleteWhereArgs(),
                    DELETE_PERMANENTLY);
            log("processMessagePart: returning false as the phone number
is blocked",
                    tracker.getMessageId());
            return false;
        }
```

```
        dispatchSmsDeliveryIntent(pdus, format, destPort,
resultReceiver,
                tracker.isClass0(), tracker.getSubId(),
tracker.getMessageId());
    }

    return true;
}

// Creates and dispatches the intent to the default SMS app, appropriate
port or via the AppSmsManager
protected void dispatchSmsDeliveryIntent(byte[][] pdus, String format,
int destPort,
        SmsBroadcastReceiver resultReceiver, boolean isClass0, int
subId, long messageId) {
// MTK-END
    Intent intent = new Intent();
    intent.putExtra("pdus", pdus);
    intent.putExtra("format", format);
    if (messageId != 0L) {
        intent.putExtra("messageId", messageId);
    }

    if (destPort == -1) {
        intent.setAction(Intents.SMS_DELIVER_ACTION);
        // Direct the intent to only the default SMS app. If we can't
find a default SMS app
        // then sent it to all broadcast receivers.
        // We are deliberately delivering to the primary user's default
SMS App.
        ComponentName componentName =
SmsApplication.getDefaultSmsApplication(mContext, true);
        if (componentName != null) {
            // Deliver SMS message only to this receiver.
            intent.setComponent(componentName);
            logWithLocalLog("Delivering SMS to: " +
componentName.getPackageName()
                    + " " + componentName.getClassName(), messageId);
        } else {
            intent.setComponent(null);
        }

        // Handle app specific sms messages.
        AppSmsManager appManager = mPhone.getAppSmsManager();
        if (appManager.handleSmsReceivedIntent(intent)) {
            // The AppSmsManager handled this intent, we're done.
            dropSms(resultReceiver);
            return;
        }
    } else {
        intent.setAction(Intents.DATA_SMS_RECEIVED_ACTION);
        Uri uri = Uri.parse("sms://localhost:" + destPort);
        intent.setData(uri);
        intent.setComponent(null);
```

```
    }

    Bundle options = handleSmsWhitelisting(intent.getComponent(),
isClass0);
    //调用sendOrderedBroadcastAsUser发出Intents.SMS_DELIVER_ACTION广播。
    dispatchIntent(intent, android.Manifest.permission.RECEIVE_SMS,
            AppOpsManager.OPSTR_RECEIVE_SMS, options, resultReceiver,
UserHandle.SYSTEM, subId);
    }
}
```

7. InboundSmsHandler的SmsBroadcastReceiver接收Intents.SMS_DELIVER_ACTION广播

```
public void onReceive(Context context, Intent intent) {
            handleAction(intent, true);
}
private synchronized void handleAction(Intent intent, boolean onReceive)
{
    String action = intent.getAction();
    int subId =
intent.getIntExtra(SubscriptionManager.EXTRA_SUBSCRIPTION_INDEX,
            SubscriptionManager.INVALID_SUBSCRIPTION_ID);
    if (action.equals(Intents.SMS_DELIVER_ACTION)) {
        // Now dispatch the notification only intent
        //将Intents.SMS_DELIVER_ACTION转换为Intents.SMS_RECEIVED_ACTION
        intent.setAction(Intents.SMS_RECEIVED_ACTION);
        // Allow registered broadcast receivers to get this intent even
        // when they are in the background.
        intent.setComponent(null);
        // All running users will be notified of the received sms.
        Bundle options = handleSmsWhitelisting(null, false /*
bgActivityStartAllowed */);

        setWaitingForIntent(intent);
        //继续调用dispatchIntent发出第二次广播
        dispatchIntent(intent, android.Manifest.permission.RECEIVE_SMS,
                AppOpsManager.OPSTR_RECEIVE_SMS,
                options, this, UserHandle.ALL, subId);
    }
}
```

8. 在android/packages/apps/Messaging/AndroidManifest.xml中SmsReceiver接收Intents.SMS_RECEIVED_ACTION广播

- ```
  <receiver android:name=".receiver.SmsReceiver"
            android:enabled="false"
            android:exported="true"
            android:permission="android.permission.BROADCAST_SMS">
      <intent-filter android:priority="2147483647">
          <action android:name="android.provider.Telephony.SMS_RECEIVED"
  />
      </intent-filter>
      <intent-filter android:priority="2147483647">
          <action android:name="android.provider.Telephony.MMS_DOWNLOADED"
  />
      </intent-filter>
  </receiver>
  ```

- SmsReceiver位于 packages/apps/Messaging/src/com/android/messaging/receiver/SmsReceiver.java

- ```
  public void onReceive(final Context context, final Intent intent) {
      LogUtil.v(TAG, "SmsReceiver.onReceive " + intent);
      // On KLP+ we only take delivery of SMS messages in
  SmsDeliverReceiver.
      if (PhoneUtils.getDefault().isSmsEnabled()) {
          final String action = intent.getAction();
          if (OsUtil.isSecondaryUser() &&

  (Telephony.Sms.Intents.SMS_RECEIVED_ACTION.equals(action) ||
                          // TODO: update this with the actual constant
  from Telephony

   "android.provider.Telephony.MMS_DOWNLOADED".equals(action))) {
              postNewMessageSecondaryUserNotification();
          } else if (!OsUtil.isAtLeastKLP()) {
              deliverSmsIntent(context, intent);
          }
      }
  }

  public static void deliverSmsIntent(final Context context, final Intent
  intent) {
      //首先在intent 对象中获取pdu 和format 两个信息，最后调用
  SmsMessage.createFromPdu(pdu, format)创建SmsMessage对象。
      final android.telephony.SmsMessage[] messages =
  getMessagesFromIntent(intent);

      // Check messages for validity
      if (messages == null || messages.length < 1) {
          LogUtil.e(TAG, "processReceivedSms: null or zero or ignored
  message");
          return;
      }

      final int errorCode =
              intent.getIntExtra(EXTRA_ERROR_CODE,
  SendStatusReceiver.NO_ERROR_CODE);
      // Always convert negative subIds into -1
  ```

```
    int subId =
PhoneUtils.getDefault().getEffectiveIncomingSubIdFromSystem(
            intent, EXTRA_SUB_ID);
    deliverSmsMessages(context, subId, errorCode, messages);
    if (MmsUtils.isDumpSmsEnabled()) {
        final String format = intent.getStringExtra("format");
        DebugUtils.dumpSms(messages[0].getTimestampMillis(), messages,
format);
    }
}

public static void deliverSmsMessages(final Context context, final int
subId,
        final int errorCode, final android.telephony.SmsMessage[]
messages) {
    final ContentValues messageValues =
            MmsUtils.parseReceivedSmsMessage(context, messages,
errorCode);

    LogUtil.v(TAG, "SmsReceiver.deliverSmsMessages");

    final long nowInMillis =  System.currentTimeMillis();
    final long receivedTimestampMs =
MmsUtils.getMessageDate(messages[0], nowInMillis);

    messageValues.put(Sms.Inbox.DATE, receivedTimestampMs);
    // Default to unread and unseen for us but ReceiveSmsMessageAction
will override
    // seen for the telephony db.
    messageValues.put(Sms.Inbox.READ, 0);
    messageValues.put(Sms.Inbox.SEEN, 0);
    if (OsUtil.isAtLeastL_MR1()) {
        messageValues.put(Sms.SUBSCRIPTION_ID, subId);
    }

    if (messages[0].getMessageClass() ==
android.telephony.SmsMessage.MessageClass.CLASS_0 ||
            DebugUtils.debugClassZeroSmsEnabled()) {
        Factory.get().getUIIntents().launchClassZeroActivity(context,
messageValues);
    } else {
        final ReceiveSmsMessageAction action = new
ReceiveSmsMessageAction(messageValues);
        //在后台激活executeAction()异步任务
        action.start();
    }
}
```

9. ReceiveSmsMessageAction的executeAction()方法将新短信保存到数据库并通过Notification显示短信通知

    ```
    protected Object executeAction() {
        final Context context = Factory.get().getApplicationContext();
        final ContentValues messageValues =
    actionParameters.getParcelable(KEY_MESSAGE_VALUES);
    ```

```
    final DatabaseWrapper db = DataModel.get().getDatabase();

    // Get the SIM subscription ID
    Integer subId = messageValues.getAsInteger(Sms.SUBSCRIPTION_ID);
    // already in the conversation.
    if (!OsUtil.isSecondaryUser()) {
        final String text = messageValues.getAsString(Sms.BODY);
        final String subject = messageValues.getAsString(Sms.SUBJECT);
        final long sent = messageValues.getAsLong(Sms.DATE_SENT);
        final ParticipantData self =
ParticipantData.getSelfParticipant(subId);
        final Integer pathPresent =
messageValues.getAsInteger(Sms.REPLY_PATH_PRESENT);
        final String smsServiceCenter =
messageValues.getAsString(Sms.SERVICE_CENTER);
        String conversationServiceCenter = null;
        // Only set service center if message REPLY_PATH_PRESENT = 1
        if (pathPresent != null && pathPresent == 1 &&
!TextUtils.isEmpty(smsServiceCenter)) {
            conversationServiceCenter = smsServiceCenter;
        }

        db.beginTransaction();
        try {
            final String participantId =

 BugleDatabaseOperations.getOrCreateParticipantInTransaction(db,
rawSender);
            final String selfId =

 BugleDatabaseOperations.getOrCreateParticipantInTransaction(db, self);

            message = MessageData.createReceivedSmsMessage(messageUri,
conversationId,
                    participantId, selfId, text, subject, sent,
received, seen, read);
             //将新短信存入数据库
            BugleDatabaseOperations.insertNewMessageInTransaction(db,
message);
            //更新短信会话消息

 BugleDatabaseOperations.updateConversationMetadataInTransaction(db,
conversationId,
                    message.getMessageId(),
message.getReceivedTimeStamp(), blocked,
                    conversationServiceCenter, true /*
shouldAutoSwitchSelfId */);
            SyncManager.immediateSync();//TINNO ADD FOR UDCFAA-1295 BY
HONGGJIANG.XIAO 20200611
            final ParticipantData sender = ParticipantData.getFromId(db,
participantId);
            BugleActionToasts.onMessageReceived(conversationId, sender,
message);
            db.setTransactionSuccessful();
        } finally {
```

```
            db.endTransaction();
        }
        LogUtil.i(TAG, "ReceiveSmsMessageAction: Received SMS message "
+ message.getMessageId()
                + " in conversation " + message.getConversationId()
                + ", uri = " + messageUri);

        actionParameters.putInt(KEY_SUB_ID, subId);

 ProcessPendingMessagesAction.scheduleProcessPendingMessagesAction(false
, this);
    } else {
        if (LogUtil.isLoggable(TAG, LogUtil.DEBUG)) {
            LogUtil.d(TAG, "ReceiveSmsMessageAction: Not inserting
received SMS message for "
                    + "secondary user.");
        }
    }
    // Show a notification to let the user know a new message has
arrived
    BugleNotifications.update(false/*silent*/, conversationId,
BugleNotifications.UPDATE_ALL);

    MessagingContentProvider.notifyMessagesChanged(conversationId);
    MessagingContentProvider.notifyPartsChanged();

    return message;
}
```