简述

- 结束发送流程追踪和解决拉取代码的一些问题。

发送流程

app

1. 打开短信应用会进入ConversationListActivity，位于packages/apps/Messaging/src/com/android/messaging/ui/conversationlist/ConversationlistActivity.java
2. 创建新的消息，进入ConversationActivity，位于ui/conversation/ConversationActivity.java
    - 在onCreate()中调用updateUiState()创建ConversationFragment对象
3. 在ConversationFragment的onCreateView获得ComposeMessageView并bind
    - ComposeMessageView：This view contains the UI required to generate and send messages.
    - ```
      public View onCreateView(final LayoutInflater inflater, final ViewGroup container,final Bundle savedInstanceState) {
           mComposeMessageView = (ComposeMessageView)view.findViewById(R.id.message_compose_view_container);
           // Bind the compose message view to the DraftMessageData

      mComposeMessageView.bind(DataModel.get().createDraftMessageData(mBinding.getData().getConversationId()), this);
      }
      ```
4. 在ComposeMessageView的onFinishInflate()中获取发送按钮并绑定点击事件
    - ```
      private ImageButton mSendButton;
      protected void onFinishInflate() {
          mSendButton = (ImageButton) findViewById(R.id.send_message_button);
          mSendButton.setOnClickListener(new OnClickListener() {
              @Override
              public void onClick(final View clickView) {
                  sendMessageInternal(true /* checkMessageSize */);
              }
          });
      }
      ```
    - 在sendMessageInternal（）中设置短信的内容和接收者，检查短信格式，根据结果执行相对应的操作
        - ```
          //ConversationFragment实现了IComposeMessageViewHost
          private IComposeMessageViewHost mHost;
          private void sendMessageInternal(final boolean checkMessageSize) {
                LogUtil.i(LogUtil.BUGLE_TAG, "UI initiated message sending in conversation " +
                        mBinding.getData().getConversationId());
          ```
          ```

```
      // Check the host for pre-conditions about any action.
      if (mHost.isReadyForAction()) {
          mInputManager.showHideSimSelector(false /* show */, true
/* animate */);
          //mBinding.getData()获得DraftMessageData对象，设置短信的内容
          final String messageToSend =
mComposeEditText.getText().toString();
          mBinding.getData().setMessageText(messageToSend);
          //设置短信的接收者
          final String subject =
mComposeSubjectText.getText().toString();
          mBinding.getData().setMessageSubject(subject);
          // Asynchronously check the draft against various
requirements before sending.
          mBinding.getData().checkDraftForAction(checkMessageSize,
                  mHost.getConversationSelfSubId(), new
CheckDraftTaskCallback() {
              @Override
              public void onDraftChecked(DraftMessageData data, int
result) {
                  mBinding.ensureBound(data);
                  switch (result) {
                      case CheckDraftForSendTask.RESULT_PASSED:
                          // Continue sending after check succeeded.
                          //将DraftMessageData对象变为MessageData对象
                          final MessageData message =
mBinding.getData().prepareMessageForSending(mBinding);
                          if (message != null &&
message.hasContent()) {
                              playSentSound();
                              //使用ConversationFragment类的
sendMessage()方法开始发送Message。
                              mHost.sendMessage(message);
                              hideSubjectEditor();
                              if
(AccessibilityUtil.isTouchExplorationEnabled(getContext())) {

 AccessibilityUtil.announceForAccessibilityCompat(
                                      ComposeMessageView.this,
null,
                                      R.string.sending_message);
                              }
                          }
                          break;
                  }
              }
          }, mBinding);
      }
  }
```

```
/*首先创建DraftMessageData类的内部类对象CheckDraftForSendTask,它继承了SafeAsync Task;接着调用此对象的executeOnThreadPool方法触发重写父类的三个方法调用onPreExecute、dolnBackgroundTimed 和onPostExecute,这几个方法的处理逻辑是发送短信的前置条件判断，最终通过mCallback.onDraftChecked调用将判断结果发送给CheckDraftTaskCallback对象。*/
public void checkDraftForAction(final boolean checkMessageSize,
final int selfSubId,final CheckDraftTaskCallback callback,
final Binding<DraftMessageData> binding) {
     new CheckDraftForSendTask(checkMessageSize, selfSubId,
callback, binding).executeOnThreadPool((Void) null);
}
```

5. 在ConversationFragment类的sendMessage()中，调用ConversationData的sendMessage()方法

```
public void sendMessage(final MessageData message) {
    if (isReadyForAction()) {
        if (ensureKnownRecipients()) {
            // Merge the caption text from attachments into the text
body of the messages
            message.consolidateText();
            //获得ConversationData，调用它的sendMessage()方法
            mBinding.getData().sendMessage(mBinding, message);
            mComposeMessageView.resetMediaPickerState();
        }
    }
}
```

6. 在ConversationData的sendMessage()中判断并根据用户的api和sub执行对应的insertNewMessage()方法。

```
public void sendMessage(final BindingBase<ConversationData>
binding,final MessageData message) {
    //boolean isAtLeastL_MR1(): True if the version of Android that
we're running on is at least L MR1(API level 22)
    if (!OsUtil.isAtLeastL_MR1() || message.getSelfId() == null) {
            InsertNewMessageAction.insertNewMessage(message);
        } else {
            final int systemDefaultSubId =
PhoneUtils.getDefault().getDefaultSmsSubscriptionId();
            if (systemDefaultSubId !=
ParticipantData.DEFAULT_SELF_SUB_ID &&

 mSelfParticipantsData.isDefaultSelf(message.getSelfId())) {
                // Lock the sub selection to the system default SIM as
soon as the user clicks on
                // the send button to avoid races between this and when
InsertNewMessageAction is
                // actually executed on the data model thread, during
which the user can potentially
                // change the system default SIM in Settings.
                InsertNewMessageAction.insertNewMessage(message,
systemDefaultSubId);
            } else {
                InsertNewMessageAction.insertNewMessage(message);
```

```
            }
        }
```

7. 在InsertNewMessageAction中执行insertNewMessage()

   - InsertNewMessageAction used to **convert a draft message to an outgoing message**. Its writes SMS messages to the telephony db, but SendMessageAction is responsible for inserting MMS message into the telephony DB. The latter also does the actual sending of the message in the background.The latter is also responsible for re-sending a failed message.

   - ```
     //Insert message (no listener)
     public static void insertNewMessage(final MessageData message) {
         final InsertNewMessageAction action = new
     InsertNewMessageAction(message);
         action.start();
     }

     //根据Action 的处理机制，将以后台异步方式调用InsertNewMessageAction对象的
     executeAction().使用了Android的IntentService运行机制
     protected Object executeAction() {
         //...先将Message插入到数据库
         ...

      ProcessPendingMessagesAction.scheduleProcessPendingMessagesAction(false
     , this);
         return message;
     }
     ```

8. 在ProcessPendingMessagesAction的scheduleProcessPendingMessagesAction()中

   - ```
     public static void scheduleProcessPendingMessagesAction(final boolean
     failed,final Action processingAction) {
         LogUtil.i(TAG, "ProcessPendingMessagesAction: Scheduling pending
     messages"+(failed ? "(message failed)" : ""));
         // Can safely clear any pending alarms or connectivity events as
     either an action
         // is currently running or we will run now or register if pending
     actions possible.
         unregister();

         final boolean isDefaultSmsApp =
     PhoneUtils.getDefault().isDefaultSmsApp();
         boolean scheduleAlarm = false;
         // If message succeeded and if Bugle is default SMS app just carry
     on with next message
         if (!failed && isDefaultSmsApp) {
             // Clear retry attempt count as something just succeeded
             setRetry(0);

             // Lookup and queue next message for immediate processing by
     background worker
             //  iff there are no pending messages this will do nothing and
     return true.
     ```

```
        final ProcessPendingMessagesAction action = new
ProcessPendingMessagesAction();
        if (action.queueActions(processingAction)) {
            if (LogUtil.isLoggable(TAG, LogUtil.VERBOSE)) {
                if (processingAction.hasBackgroundActions()) {
                    LogUtil.v(TAG, "ProcessPendingMessagesAction: Action
queued");
                } else {
                    LogUtil.v(TAG, "ProcessPendingMessagesAction: No
actions to queue");
                }
            }
            // Have queued next action if needed, nothing more to do
            return;
        }
        // In case of error queuing schedule a retry
        scheduleAlarm = true;
        LogUtil.w(TAG, "ProcessPendingMessagesAction: Action failed to
queue; retrying");
    }
}
```

```
/**
Queue any pending actions
return true if action queued (or no actions to queue) else false
*/
private boolean queueActions(final Action processingAction) {
      final DatabaseWrapper db = DataModel.get().getDatabase();
      final long now = System.currentTimeMillis();
      boolean succeeded = true;
      final int subId =
processingAction.actionParameters.getInt(KEY_SUB_ID,
ParticipantData.DEFAULT_SELF_SUB_ID);

      LogUtil.i(TAG, "ProcessPendingMessagesAction: Start queueing for
subId " + subId);

      final String selfId = ParticipantData.getParticipantId(db, subId);
      if (selfId == null) {
          // This could be happened before refreshing participant.
          LogUtil.w(TAG, "ProcessPendingMessagesAction: selfId is
null");
          return false;
      }
      // Will queue no more than one message to send plus one message to
download
      // This keeps outgoing messages "in order" but allow downloads to
happen even if sending
      // gets blocked until messages time out. Manual resend bumps
messages to head of queue.
      final String toSendMessageId = findNextMessageToSend(db, now,
selfId);
      final String toDownloadMessageId = findNextMessageToDownload(db,
now, selfId);
      if (toSendMessageId != null) {
```

```
            LogUtil.i(TAG, "ProcessPendingMessagesAction: Queueing message
" + toSendMessageId
                    + " for sending");
            // This could queue nothing
            if
(!SendMessageAction.queueForSendInBackground(toSendMessageId,
processingAction)) {
                LogUtil.w(TAG, "ProcessPendingMessagesAction: Failed to
queue message "
                        + toSendMessageId + " for sending");
                succeeded = false;
            }
        }
}
```

9. 在SendMessageAction的queueForSendInBackground()中

- Action used to send an outgoing message. It writes MMS messages to the telephony db.InsertNewMessageAction writes SMS messages to the telephony db). It also initiates the actual sending. It will all be used for re-sending a failed message.
- ```
  //Queue sending of existing message (can only be called during execute
  of action)
  static boolean queueForSendInBackground(final String messageId,final
  Action processingAction) {
      final SendMessageAction action = new SendMessageAction();
      return action.queueAction(messageId, processingAction);
  }

  //Read message from database and queue actual sending
  private boolean queueAction(final String messageId, final Action
  processingAction) {
      //从数据库中读取Message，如果数据库没有就不用发送了
      ...
      //根据读取到的协议是不是sms的
      final boolean isSms = (message.getProtocol() ==
  MessageData.PROTOCOL_SMS);
      if (isSms) {
          //获得服务中心
          //BugleDatabaseOperations:manages updating our local database
          final String smsc =
  BugleDatabaseOperations.getSmsServiceCenterForConversation(db,
  conversationId);
          actionParameters.putString(KEY_SMS_SERVICE_CENTER, smsc);

          if (recipients.size() == 1) {
              final String recipient = recipients.get(0);

              actionParameters.putString(KEY_RECIPIENT, recipient);
              // Queue actual sending for SMS
              //Queues up background actions for background processing
  after the current action has completed its processing
              //将当前的发送短信的业务添加到队列中，轮到这条短信时会调用
  SendMessageAction 对象的doBackgroundWork()在后台执行耗时的异步任务
              processingAction.requestBackgroundWork(this);
  ```
  ```

```
            if (LogUtil.isLoggable(TAG, LogUtil.DEBUG)) {
                LogUtil.d(TAG, "SendMessageAction: Queued SMS message "
+ messageId
                        + " for sending");
            }
            return true;
        }
    }
}

//Do work in a long running background worker
thread.requestBackgroundWork() needs to be called for this method to be
called
protected Bundle doBackgroundWork() {
    final MessageData message =
actionParameters.getParcelable(KEY_MESSAGE);
    final String messageId = actionParameters.getString(KEY_MESSAGE_ID);
    Uri messageUri = actionParameters.getParcelable(KEY_MESSAGE_URI);
    Uri updatedMessageUri = null;
    final boolean isSms = message.getProtocol() ==
MessageData.PROTOCOL_SMS;
    final int subId = actionParameters.getInt(KEY_SUB_ID,
ParticipantData.DEFAULT_SELF_SUB_ID);
    final String subPhoneNumber =
actionParameters.getString(KEY_SUB_PHONE_NUMBER);

    LogUtil.i(TAG, "SendMessageAction: Sending " + (isSms ? "SMS" :
"MMS") + " message "
            + messageId + " in conversation " +
message.getConversationId());

    int status;
    int rawStatus = MessageData.RAW_TELEPHONY_STATUS_UNDEFINED;
    int resultCode = MessageData.UNKNOWN_RESULT_CODE;
    if (isSms) {
        Assert.notNull(messageUri);
        final String recipient =
actionParameters.getString(KEY_RECIPIENT);
        final String messageText = message.getMessageText();
        final String smsServiceCenter =
actionParameters.getString(KEY_SMS_SERVICE_CENTER);
        final boolean deliveryReportRequired =
MmsUtils.isDeliveryReportRequired(subId);

        status = MmsUtils.sendSmsMessage(recipient, messageText,
messageUri, subId,
                smsServiceCenter, deliveryReportRequired);
    }
}
```

10. 在MmsUtils的sendSmsMessage()中
    - MmsUtils：Utils for sending sms/mms messages.
    - ```
      public static int sendSmsMessage(final String recipient, final String
      messageText,final Uri requestUri, final int subId,
      ```
      ```

```
            final String smsServiceCenter, final boolean
requireDeliveryReport) {
        final Context context = Factory.get().getApplicationContext();
        int status = MMS_REQUEST_MANUAL_RETRY;
        try {
            // Send a single message
            final SendResult result = SmsSender.sendMessage(
                    context,
                    subId,
                    recipient,
                    messageText,
                    smsServiceCenter,
                    requireDeliveryReport,
                    requestUri);
        } catch (final Exception e) {
            LogUtil.e(TAG, "MmsUtils: failed to send SMS " + e, e);
        }
        return status;
    }
```

11. 在SmsSender的sendMessage()中
    - SmsSender：Class that sends chat message via SMS
    - ```
      public static SendResult sendMessage(final Context context,  final int
      subId, String dest,String message, final String serviceCenter, final
      boolean requireDeliveryReport,final Uri messageUri) throws SmsException
      {
          // Divide the input message by SMS length limit
          final SmsManager smsManager = PhoneUtils.get(subId).getSmsManager();
          final ArrayList<String> messages =
      smsManager.divideMessage(message);
          if (messages == null || messages.size() < 1) {
              throw new SmsException("SmsSender: fails to divide message");
          }
          // Prepare the send result, which collects the send status for each
      part
          final SendResult pendingResult = new SendResult(messages.size());
          sPendingMessageMap.put(messageUri, pendingResult);
          // Actually send the sms
          sendInternal(context, subId, dest, messages, serviceCenter,
      requireDeliveryReport, messageUri);
      }

      // Actually sending the message using SmsManager
      private static void sendInternal(final Context context, final int subId,
      String dest,final ArrayList<String> messages, final String
      serviceCenter,final boolean requireDeliveryReport, final Uri messageUri)
      throws SmsException {
          Assert.notNull(context);
          final SmsManager smsManager = PhoneUtils.get(subId).getSmsManager();
          final int messageCount = messages.size();
          final ArrayList<PendingIntent> deliveryIntents = new
      ArrayList<PendingIntent>(messageCount);
          final ArrayList<PendingIntent> sentIntents = new
      ArrayList<PendingIntent>(messageCount);
      ```
      ```

```
    for (int i = 0; i < messageCount; i++) {
        // Make pending intents different for each message part
        final int partId = (messageCount <= 1 ? 0 : i + 1);
        if (requireDeliveryReport && (i == (messageCount - 1))) {
            // only care about the delivery status of the last part
            //如果是最后一个部分使用deliveryIntents，
MESSAGE_DELIVERED_ACTION
            deliveryIntents.add(PendingIntent.getBroadcast(
                    context,
                    partId,
                    getSendStatusIntent(context,
SendStatusReceiver.MESSAGE_DELIVERED_ACTION,
                            messageUri, partId, subId),
                    0/*flag*/));
        } else {
            deliveryIntents.add(null);
        }
        //如果不是最后一个部分，使用sentIntents，MESSAGE_SENT_ACTION
        sentIntents.add(PendingIntent.getBroadcast(
                context,
                partId,
                getSendStatusIntent(context,
SendStatusReceiver.MESSAGE_SENT_ACTION,
                        messageUri, partId, subId),
                0/*flag*/));
    }
    if (sSendMultipartSmsAsSeparateMessages == null) {
        sSendMultipartSmsAsSeparateMessages = MmsConfig.get(subId)
                .getSendMultipartSmsAsSeparateMessages();
    }
    try {
        if (sSendMultipartSmsAsSeparateMessages) {
            // If multipart sms is not supported, send them as separate
messages
            for (int i = 0; i < messageCount; i++) {
                smsManager.sendTextMessage(dest,
                        serviceCenter,
                        messages.get(i),
                        sentIntents.get(i),
                        deliveryIntents.get(i));
            }
        } else {
            smsManager.sendMultipartTextMessage(
                    dest, serviceCenter, messages, sentIntents,
deliveryIntents);
        }
    } catch (final Exception e) {
        throw new SmsException("SmsSender: caught exception in sending "
+ e);
    }
}
```

Framework

1. SmsManager的sendTextMessage()中
    - SmsManager位于 frameworks/base/telephony/java/android/telephony/java/android/telephony/SmsManager.java
    - ```
      //Send a text based SMS
      public void sendTextMessage(String destinationAddress, String scAddress,
      String text,PendingIntent sentIntent, PendingIntent deliveryIntent) {
          sendTextMessageInternal(destinationAddress, scAddress, text,
      sentIntent, deliveryIntent,
                  true /* persistMessage*/);
      }
      private void sendTextMessageInternal(String destinationAddress, String
      scAddress,
              String text, PendingIntent sentIntent, PendingIntent
      deliveryIntent,
              boolean persistMessage) {
          try {
              //SmsController继承了ISmsImplBase，而ISmsImplBase继承了ISms.Stub
              ISms iccISms = getISmsServiceOrThrow();
              iccISms.sendTextForSubscriber(getSubscriptionId(),
      ActivityThread.currentPackageName(),
                      destinationAddress,
                      scAddress, text, sentIntent, deliveryIntent,
                      persistMessage);
          } catch (RemoteException ex) {
              // ignore it
          }
      }
      ```

2. SmsController的sendTextForSubscriber()中
    - 跨进程调用，从Messaging进程到phone进程
    - ISms.aidl位于 frameworks/base/telephony/java/com/android/internal/telephony/ISms.aidl
    - UiccSmsController位于 frameworks/opt/telephony/java/com/android/internal/telephony/SmsController.java
    - ```
      public void sendTextForSubscriber(int subId, String
      callingPackage,String callingAttributionTag, String destAddr, String
      scAddr, String text,PendingIntent sentIntent, PendingIntent
      deliveryIntent,boolean persistMessageForNonDefaultSmsApp, long
      messageId) {
            if (callingPackage == null) {
                callingPackage = getCallingPackage();
            }
            if
      (!getSmsPermissions(subId).checkCallingCanSendText(persistMessageForNonD
      efaultSmsApp,
                    callingPackage, callingAttributionTag, "Sending SMS
      message")) {
      ```

```
            sendErrorInPendingIntent(sentIntent,
SmsManager.RESULT_ERROR_GENERIC_FAILURE);
            return;
        }
        long token = Binder.clearCallingIdentity();
        //获得发送者的信息
        SubscriptionInfo info;
        try {
            info = getSubscriptionInfo(subId);
        } finally {
            Binder.restoreCallingIdentity(token);
        }
        if (isBluetoothSubscription(info)) {
            sendBluetoothText(info, destAddr, text, sentIntent,
deliveryIntent);
        } else {
            //关键步骤
            sendIccText(subId, callingPackage, destAddr, scAddr, text,
sentIntent, deliveryIntent,
                    persistMessageForNonDefaultSmsApp, messageId);
        }
}
private void sendIccText(int subId, String callingPackage, String
destAddr,
                String scAddr, String text, PendingIntent sentIntent,
PendingIntent deliveryIntent,
                boolean persistMessageForNonDefaultSmsApp, long messageId)
{
        Rlog.d(LOG_TAG, "sendTextForSubscriber iccSmsIntMgr"+ "
Subscription: " + subId + " id: " + messageId);
        IccSmsInterfaceManager iccSmsIntMgr =
getIccSmsInterfaceManager(subId);
        if (iccSmsIntMgr != null) {
            //关键步骤
            iccSmsIntMgr.sendText(callingPackage, destAddr, scAddr, text,
sentIntent,
                    deliveryIntent, persistMessageForNonDefaultSmsApp,
messageId);
        } else {
            Rlog.e(LOG_TAG, "sendTextForSubscriber iccSmsIntMgr is null
for"
                    + " Subscription: " + subId + " id: " + messageId);
            sendErrorInPendingIntent(sentIntent,
SmsManager.RESULT_ERROR_GENERIC_FAILURE);
        }
}
```

3. 在IccSmsInterfaceManager的sendText()中

   - IccSmsInterfaceManager to provide an inter-process communication to access Sms in Icc
   - Uicc:The universal integrated circuit card (UICC) is the smart card (integrated circuit card) used in mobile terminals in GSM and UMTS networks. The UICC ensures the integrity and security of all kinds of personal data, and it typically holds a few hundred kilobytes. The UICC's primary component is a SIM card.

- UMTS：Universal Mobile Telecommunications System，a 3g technology

```
public void sendText(String callingPackage, String destAddr, String
scAddr,String text, PendingIntent sentIntent, PendingIntent
deliveryIntent,boolean persistMessageForNonDefaultSmsApp) {
    //A permissions check
    //This method checks only if the calling package has the permission
to send the sms.

 mPhone.getContext().enforceCallingPermission(Manifest.permission.SEND_S
MS,"Sending SMS message");
    //关键步骤
    sendTextInternal(callingPackage, destAddr, scAddr, text, sentIntent,
deliveryIntent,
        persistMessageForNonDefaultSmsApp);
}

//Send a text based SMS
public SmsDispatchersController mDispatchersController;
private void sendTextInternal(String callingPackage, String destAddr,
String scAddr,String text, PendingIntent sentIntent, PendingIntent
deliveryIntent,boolean persistMessageForNonDefaultSmsApp, int priority,
boolean expectMore,
int validityPeriod, boolean isForVvm, long messageId) {
      if (Rlog.isLoggable("SMS", Log.VERBOSE)) {
          log("sendText: destAddr=" + destAddr + " scAddr=" + scAddr
                  + " text='" + text + "' sentIntent=" + sentIntent + "
deliveryIntent="
                  + deliveryIntent + " priority=" + priority + "
expectMore=" + expectMore
                  + " validityPeriod=" + validityPeriod + " isForVVM=" +
isForVvm
                  + " " +
SmsController.formatCrossStackMessageId(messageId));
      }
      notifyIfOutgoingEmergencySms(destAddr);
      destAddr = filterDestAddress(destAddr);
      //关键步骤
      mDispatchersController.sendText(destAddr, scAddr, text,
sentIntent, deliveryIntent,
              null/*messageUri*/, callingPackage,
persistMessageForNonDefaultSmsApp,
              priority, expectMore, validityPeriod, isForVvm,
messageId);
    }
```

4. 在SmsDispatchersController的sendText（）方法中

```
//在SmsDispatchersController的构造方法中mCdmaDispatcher = new
CdmaSMSDispatcher(phone, this);
//在SmsDispatchersController的构造方法中mGsmDispatcher = new
GsmSMSDispatcher(phone, this, mGsmInboundSmsHandler);
private SMSDispatcher mCdmaDispatcher;mCdmaDispatcher = new
GsmSMSDispatcher()
private SMSDispatcher mGsmDispatcher;//在SmsDispatchersController的构造方
法中mGsmDispatcher = new GsmSMSDispatcher()
```

```
private ImsSmsDispatcher mImsSmsDispatcher;

public void sendText(String destAddr, String scAddr, String text,
PendingIntent sentIntent,
        PendingIntent deliveryIntent, Uri messageUri, String callingPkg,
boolean persistMessage,
        int priority, boolean expectMore, int validityPeriod, boolean
isForVvm,
        long messageId) {
    //根据不同的情况将sms分发掉
    if (mImsSmsDispatcher.isAvailable() ||
mImsSmsDispatcher.isEmergencySmsSupport(destAddr)) {
        mImsSmsDispatcher.sendText(destAddr, scAddr, text, sentIntent,
deliveryIntent,
                messageUri, callingPkg, persistMessage, priority, false
/z*expectMore*/,
                validityPeriod, isForVvm, messageId);
    } else {
        if (isCdmaMo()) {
            mCdmaDispatcher.sendText(destAddr, scAddr, text, sentIntent,
deliveryIntent,messageUri, callingPkg,
persistMessage, priority, expectMore,validityPeriod, isForVvm,
messageId);
        } else {
            mGsmDispatcher.sendText(destAddr, scAddr, text, sentIntent,
deliveryIntent,
                    messageUri, callingPkg, persistMessage, priority,
expectMore,
                    validityPeriod, isForVvm, messageId);
        }
    }
 }
```

5. 在SMSDispatcher的sendText ( ) 中

```
public void sendText(String destAddr, String scAddr, String text,
                     PendingIntent sentIntent, PendingIntent
deliveryIntent, Uri messageUri,
                     String callingPkg, boolean persistMessage, int
priority,
                     boolean expectMore, int validityPeriod, boolean
isForVvm,
                     long messageId) {
    Rlog.d(TAG, "sendText id: " + messageId);
    //根据获得的信息生成对应的pdu
    SmsMessageBase.SubmitPduBase pdu = getSubmitPdu(
            scAddr, destAddr, text, (deliveryIntent != null), null,
priority, validityPeriod);
    if (pdu != null) {
        //根据获得的信息生成对应的SmsTracker
        HashMap map = getSmsTrackerMap(destAddr, scAddr, text, pdu);
        SmsTracker tracker = getSmsTracker(callingPkg, map, sentIntent,
deliveryIntent,
                getFormat(), messageUri, expectMore, text, true
/*isText*/,
```

```
                persistMessage, priority, validityPeriod, isForVvm,
messageId);

        if (!sendSmsByCarrierApp(false /* isDataSms */, tracker)) {
            sendSubmitPdu(tracker);
        }
    } else {
        Rlog.e(TAG, "SmsDispatcher.sendText(): getSubmitPdu() returned
null" + " id: "
                + messageId);
        triggerSentIntentForFailure(sentIntent);
    }
}

/** Send a single SMS PDU. */
@UnsupportedAppUsage(maxTargetSdk = Build.VERSION_CODES.R, trackingBug =
170729553)
private void sendSubmitPdu(SmsTracker tracker) {
    sendSubmitPdu(new SmsTracker[] {tracker});
}

/** Send a multi-part SMS PDU. Usually just calls sendRawPdu(). */
private void sendSubmitPdu(SmsTracker[] trackers) {
    if (shouldBlockSmsForEcbm()) {
        Rlog.d(TAG, "Block SMS in Emergency Callback mode");
        handleSmsTrackersFailure(trackers,
SmsManager.RESULT_SMS_BLOCKED_DURING_EMERGENCY,
                NO_ERROR_CODE);
    } else {
        sendRawPdu(trackers);
    }
}

//Send a single or a multi-part SMS
public void sendRawPdu(SmsTracker[] trackers) {
    //差错验证，获取包信息
    ...
    // checkDestination() returns true if the destination is not a
premium short code or the
    // sending app is approved to send to short codes. Otherwise, a
message is sent to our
    // handler with the SmsTracker to request user confirmation before
sending.
    if (checkDestination(trackers)) {
        // check for excessive outgoing SMS usage by this app
        if (!mSmsDispatchersController
                .getUsageMonitor()
                .check(appInfo.packageName, trackers.length)) {

 sendMessage(obtainMessage(EVENT_SEND_LIMIT_REACHED_CONFIRMATION,
trackers));
            return;
        }

        for (SmsTracker tracker : trackers) {
```

```
            //判断授权开关是否开启
            if
(mSmsDispatchersController.getUsageMonitor().isSmsAuthorizationEnabled()
) {
                final SmsAuthorizationCallback callback = new
SmsAuthorizationCallback() {
                    @Override
                    public void onAuthorizationResult(final boolean
accepted) {
                        if (accepted) {
                            sendSms(tracker);
                        } else {
                            tracker.onFailed(mContext,
SmsManager.RESULT_ERROR_GENERIC_FAILURE,
                                    SmsUsageMonitor.ERROR_CODE_BLOCKED);
                        }
                    }
                };

mSmsDispatchersController.getUsageMonitor().authorizeOutgoingSms(tracker
.mAppInfo,
                        tracker.mDestAddress,tracker.mFullMessageText,
callback, this);
            } else {
                //没开启直接发送
                //实际调用的是GsmSMSDispatcher的sendSms（）方法，
                sendSms(tracker);
            }
        }
    }
        //如果是打给紧急号码，启用异步的紧急服务。
    if (mTelephonyManager.isEmergencyNumber(trackers[0].mDestAddress)) {
        new AsyncEmergencyContactNotifier(mContext).execute();
    }
}
```

6. 在GsmSMSDispatcher的sendSms()中

```
protected void sendSms(SmsTracker tracker) {
    int ss = mPhone.getServiceState().getState();

    Rlog.d(TAG, "sendSms: "
            + " isIms()=" + isIms()
            + " mRetryCount=" + tracker.mRetryCount
            + " mImsRetry=" + tracker.mImsRetry
            + " mMessageRef=" + tracker.mMessageRef
            + " mUsesImsServiceForIms=" + tracker.mUsesImsServiceForIms
            + " SS=" + ss
            + " " +
SmsController.formatCrossStackMessageId(tracker.mMessageId));

    // if sms over IMS is not supported on data and voice is not
available...
    if (!isIms() && ss != ServiceState.STATE_IN_SERVICE) {
    //In 5G case only Data Rat is reported.
```

```
        if(mPhone.getServiceState().getRilDataRadioTechnology()
                != ServiceState.RIL_RADIO_TECHNOLOGY_NR) {
            tracker.onFailed(mContext, getNotInServiceError(ss),
NO_ERROR_CODE);
            return;
        }
    }
    //当发送完成，执行EVENT_SEND_SMS_COMPLETE消息回调
    Message reply = obtainMessage(EVENT_SEND_SMS_COMPLETE, tracker);
    HashMap<String, Object> map = tracker.getData();
    byte pdu[] = (byte[]) map.get("pdu");
    byte smsc[] = (byte[]) map.get("smsc");
    if (tracker.mRetryCount > 0) {
        // per TS 23.040 Section 9.2.3.6:  If TP-MTI SMS-SUBMIT (0x01)
type
        //   TP-RD (bit 2) is 1 for retry
        //   and TP-MR is set to previously failed sms TP-MR
        if (((0x01 & pdu[0]) == 0x01)) {
            pdu[0] |= 0x04; // TP-RD
            pdu[1] = (byte) tracker.mMessageRef; // TP-MR
        }
    }

    // sms over gsm is used:
    //   if sms over IMS is not supported AND
    //   this is not a retry case after sms over IMS failed
    //     indicated by mImsRetry > 0 OR
    //   this tracker uses ImsSmsDispatcher to handle SMS over IMS. This
dispatcher has received
    //     this message because the ImsSmsDispatcher has indicated that
the message needs to
    //     fall back to sending over CS.
    if (0 == tracker.mImsRetry && !isIms() ||
tracker.mUsesImsServiceForIms) {
        if (tracker.mRetryCount == 0 && tracker.mExpectMore) {
            mCi.sendSMSExpectMore(IccUtils.bytesToHexString(smsc),
                    IccUtils.bytesToHexString(pdu), reply);
        } else {
            mCi.sendSMS(IccUtils.bytesToHexString(smsc),
                    IccUtils.bytesToHexString(pdu), reply);
        }
    } else {
        mCi.sendImsGsmSms(IccUtils.bytesToHexString(smsc),
                IccUtils.bytesToHexString(pdu), tracker.mImsRetry,
                tracker.mMessageRef, reply);
        // increment it here, so in case of SMS_FAIL_RETRY over IMS
        // next retry will be sent using IMS request again.
        tracker.mImsRetry++;
    }
}
```

7. RIL的sendSMS()

    - ```
      public void sendSMS(String smscPdu, String pdu, Message result) {
          获得radio代理
      ```
      ```

```
    IRadio radioProxy = getRadioProxy(result);
    if (radioProxy != null) {
        //注册消息
        RILRequest rr = obtainRequest(RIL_REQUEST_SEND_SMS, result,
                mRILDefaultWorkSource);

        // Do not log function args for privacy
        if (RILJ_LOGD) riljLog(rr.serialString() + "> " +
requestToString(rr.mRequest));

        GsmSmsMessage msg = constructGsmSendSmsRilRequest(smscPdu, pdu);
        if (mRadioVersion.greaterOrEqual(RADIO_HAL_VERSION_1_6)) {
            try {
                android.hardware.radio.V1_6.IRadio radioProxy16 =
                        (android.hardware.radio.V1_6.IRadio) radioProxy;
                //通过rilc继续发送
                radioProxy16.sendSms_1_6(rr.mSerial, msg);
                mMetrics.writeRilSendSms(mPhoneId, rr.mSerial,
SmsSession.Event.Tech.SMS_GSM,
                        SmsSession.Event.Format.SMS_FORMAT_3GPP,
                        getOutgoingSmsMessageId(result));
            } catch (RemoteException | RuntimeException e) {
                handleRadioProxyExceptionForRR(rr, "sendSMS", e);
            }
        } else {
            try {
                radioProxy.sendSms(rr.mSerial, msg);
                mMetrics.writeRilSendSms(mPhoneId, rr.mSerial,
SmsSession.Event.Tech.SMS_GSM,
                        SmsSession.Event.Format.SMS_FORMAT_3GPP,
                        getOutgoingSmsMessageId(result));
            } catch (RemoteException | RuntimeException e) {
                handleRadioProxyExceptionForRR(rr, "sendSMS", e);
            }
        }
    }
}
```

RIL

- 根据不同的radio版本发送sms，绑定RIL_REQUEST_SEND_SMS消息回调

总结

- 根据分层的思想，不断调用下层提供的服务，合作完成短信的发送。
- 发送中不断进行差错检测，确保发送的正确性
- 多次handle的消息回调，listener的简体，异步处理发送结果的响应