**MT Call流程分析**

对于MT(来电)来讲，首先还是会由Modem接收到信息，然后发给RILD，RILD再发送给RILJ，并在RILJ的RILReceiver中 接收到并进行处理。RILJ会从RILC收到Solicited或者Unsolicited信息，并 对之进行处理之后向上反馈。本次分析的MT(来电)流程，就属于Unsolicited信息，而信息从RILJ中的RILReceiver的run方法中 开始。

通过RILReceiver接收到MT(来电)信息，RILC将相关来电信息通过socket发送给RILJ，RILReceiver接收到之后进行读取并打包进行上报。

提示：

1. 监听与回调模型的理解
2. 观察者设计模式的了解
3. 来电信息内容载体的转化
4. Tracker跟踪者的作用
5. 异步两种方式，尤其是Message中各种方法的理解
6. Modem和RILD与RILJ的消息交互模型
7. Parcel的用法和Parcelable的关系

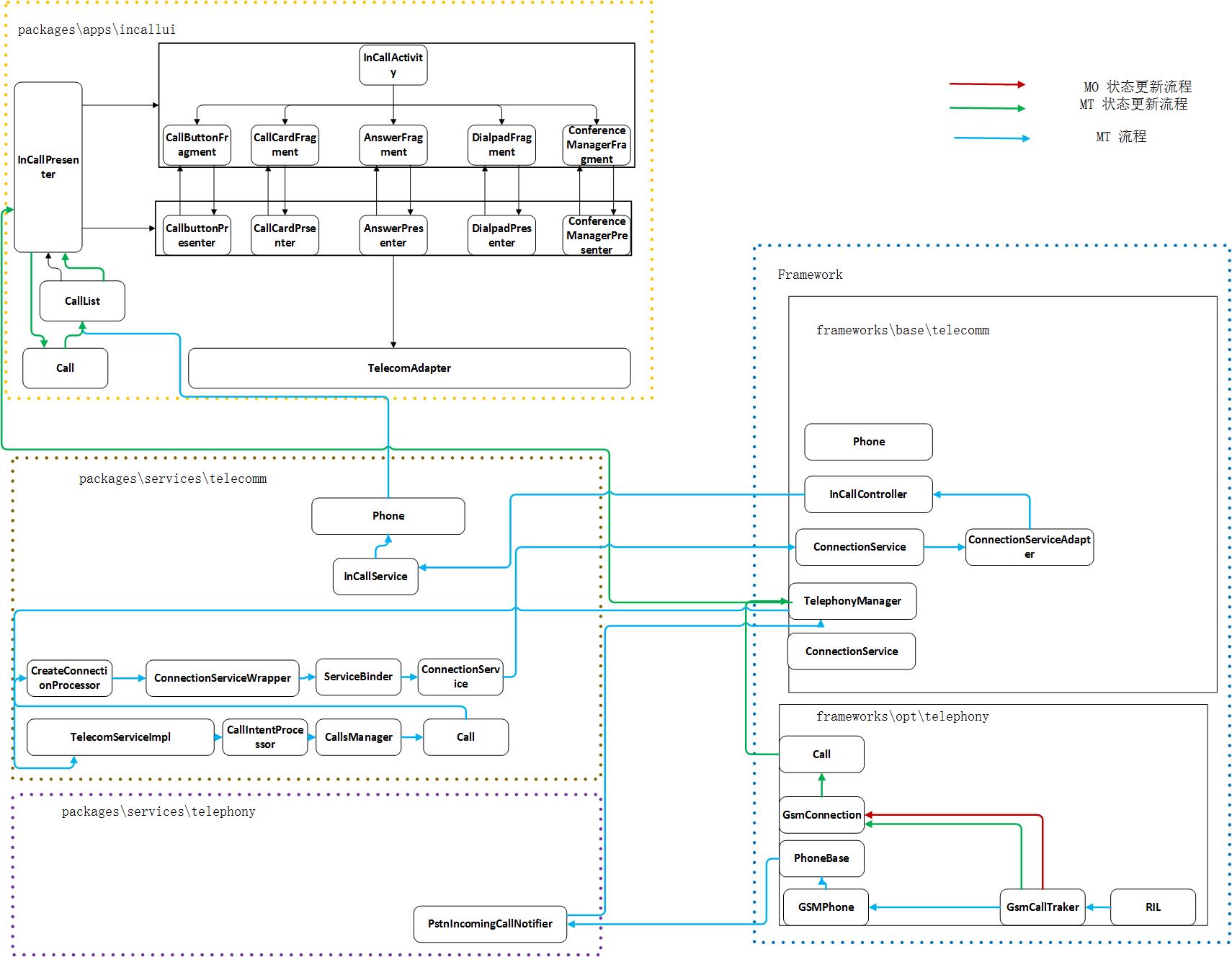
Debug提示：

1. 因为存在时间限制问题，Debug断点应该尽可能的少。
2. MTCall涉及多进程，一般确定进程名的优先级为：组件process属性 > 应用process属性 > packagename
3. 非Apk进程不适用（没有manifests.xml文件）

Log日志提示：

1. 刷Debug版本（即是是这样，也不能保证所有的Log都能打印出来）
2. 没有打出log， 但会执行的方法，可以断点调试。
3. 推荐FW\_TAG， GSMPhone, GsmCallTracker, GsmConnection, Telecom
4. App\_TAG , InCallPresenter, PrimaryCallTracker, InCall,

其整体类流程图如下：



# 

# RILD通信流程图

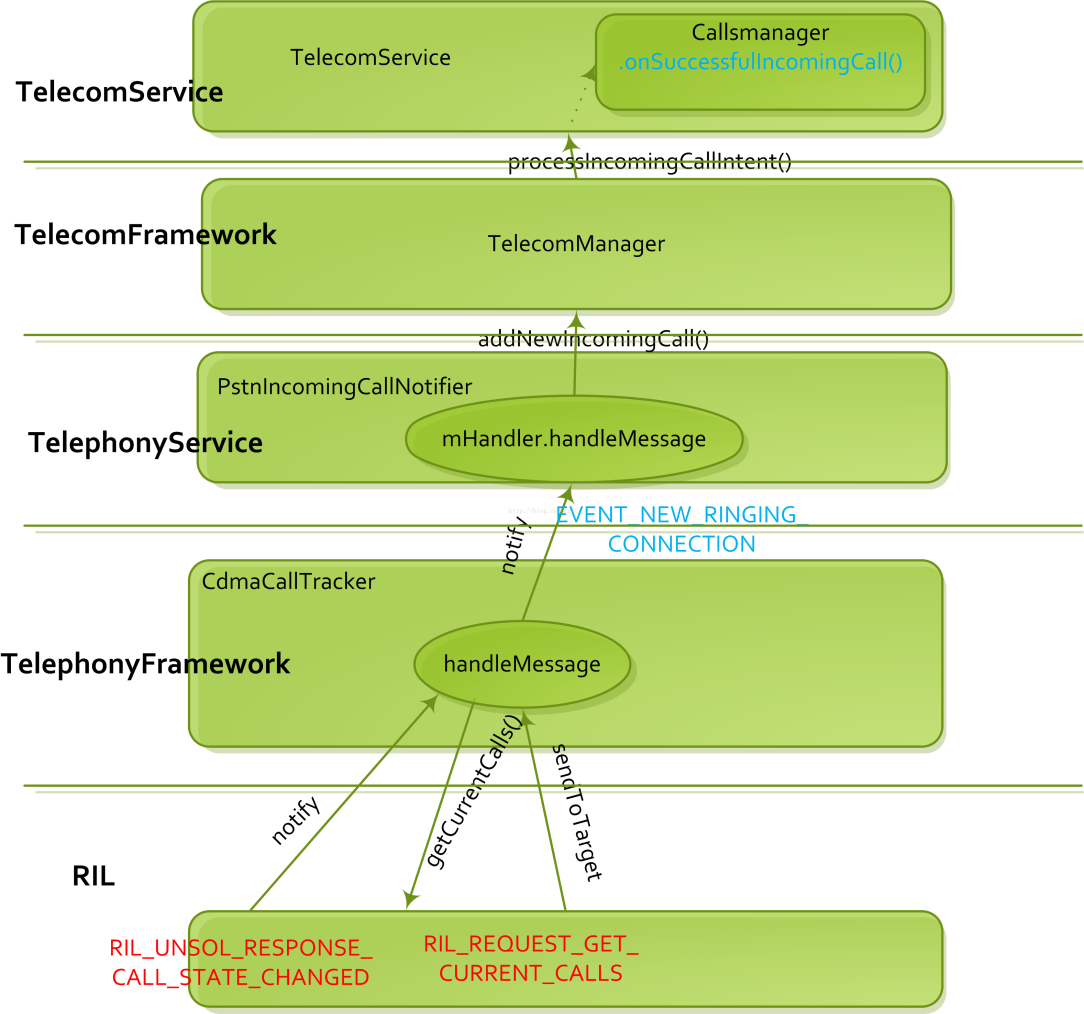
# C:\Users\xp020179\Desktop\文档\RILD与上下通信.png

# RIL内部循环

# C:\Users\xp020179\Desktop\文档\RILJ内部循环.png

# 第一部分：RIL–>GSMPhone Call状态变化 -> 发出来电通知（frameworks\opt\telephony）

其底层整体框架图为：

****

## 1、framwork/opt/telephony/…/RIL.java接收来自RILD传来的数据

首先通过RILReceiver对象接收到消息，并进行解析通过主动上报的形式将数据向上传输，接收到RIL\_UNSOL\_RESPONSE\_CALL\_STATE\_CHANGED的消息

|  |
| --- |
| private void processUnsolicited (Parcel p) {  ...  switch(response) {  ...  case RIL\_UNSOL\_RESPONSE\_CALL\_STATE\_CHANGED:  if (RILJ\_LOGD) unsljLog(response);  mCallStateRegistrants.notifyRegistrants(new AsyncResult(null, null, null));// 发出通知 |

**BaseCommands.java** 查找mCallStateRegistrants是在哪儿创建的

重点1：这其实是观察者模式的一种实现形式  
1.RefistrantList 通知者

2.Registrant 观察者，这是一个一对多的关系，在有事件更新时，凡是在名单上登记过的对象，都会收到通知。  
RegistrantList通知者支持对通知者的增加（add/addUnique）删除（remove），并且能够发出通知（notifyRegitrants）；而Registrant作为观察者，响应通知者发出的notifyRegistrant通知。整体上这个消息注册机制分为两部分，消息注册和消息通知。当调用regist方法时将Message存放进去，当其调用notify方法时将所有Message取出并发送到MessageQueue中等待处理。

## 2、framwork/opt/telephony/…GSMCallTracker.java MT Call实现

作用：GSMCallTracker在本质上是一个Handler。GSMCallTracker是Android的通话管理层。GSMCallTracker建立了ConnectionList来管理现行的通话连接，并向上层提供电话调用接口。  
查找察者被调用的地方， 两处被响应处理处理，其中一处：GsmCallTracker构造方法中通过Phone对象进行注册；另一处是CdmaCallTracker中。此处指运行GSMCallTracker

|  |
| --- |
| @Override  public void registerForCallStateChanged(Handler h, int what, Object obj) {  Registrant r = new Registrant (h, what, obj);  //添加到观察者列表  mCallStateRegistrants.add(r);  }  GsmCallTracker (GSMPhone phone) {  this.mPhone = phone;  mCi = phone.mCi;  mCi.registerForCallStateChanged(this, EVENT\_CALL\_STATE\_CHANGE, null);  mCi.registerForOn(this, EVENT\_RADIO\_AVAILABLE, null);  mCi.registerForNotAvailable(this, EVENT\_RADIO\_NOT\_AVAILABLE, null);  }  //响应处理  public void handleMessage (Message msg) {  AsyncResult ar;  if (!mPhone.mIsTheCurrentActivePhone) {  Rlog.e(LOG\_TAG, "Received message " + msg +  "[" + msg.what + "] while being destroyed. Ignoring.");  return;  }  switch (msg.what) {  …  case [EVENT\_REPOLL\_AFTER\_DELAY](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=EVENT_REPOLL_AFTER_DELAY&project=odm-foxconn-android-m-kanuti-tulip-somc):  case [EVENT\_CALL\_STATE\_CHANGE](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=EVENT_CALL_STATE_CHANGE&project=odm-foxconn-android-m-kanuti-tulip-somc):  [pollCallsWhenSafe](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=pollCallsWhenSafe&project=odm-foxconn-android-m-kanuti-tulip-somc)();  break;  case [EVENT\_RADIO\_AVAILABLE](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=EVENT_RADIO_AVAILABLE&project=odm-foxconn-android-m-kanuti-tulip-somc):  [handleRadioAvailable](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=handleRadioAvailable&project=odm-foxconn-android-m-kanuti-tulip-somc)();  break;  case [EVENT\_RADIO\_NOT\_AVAILABLE](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=EVENT_RADIO_NOT_AVAILABLE&project=odm-foxconn-android-m-kanuti-tulip-somc):  [handleRadioNotAvailable](http://opengrok.sonymobile.net/JDM-BRANCHES/xref/odm-foxconn-android-m-kanuti-tulip-somc/frameworks/opt/telephony/src/java/com/android/internal/telephony/gsm/GsmCallTracker.java#handleRadioNotAvailable)();  break;  }  } |

## 3、pollCallsWhenSafe()方法实现

pollCallsWhenSafe()方法在CallTracker.java中实现，这个 方法向底层发一个消息得到当前的Call List

|  |
| --- |
| protected void pollCallsWhenSafe() {  mNeedsPoll = true;  if (checkNoOperationsPending()) {  mLastRelevantPoll = obtainMessage(EVENT\_POLL\_CALLS\_RESULT);  mCi.getCurrentCalls(mLastRelevantPoll);// //RIL.java中的getCurrentCalls方法  }  }  回到RIL.java getCurrentCalls 将RIL\_REQUEST\_GET\_CURRENT\_CALLS 消息封装成RILRequest类型并发送。  public void getCurrentCalls(Message result) {  RILRequest rr = RILRequest.obtain(RIL\_REQUEST\_GET\_CURRENT\_CALLS, result);  if (RILJ\_LOGD)  riljLog(rr.serialString() + "> " + requestToString(rr.mRequest));  send(rr);  } |

RIL.java 有三处接收处理RIL\_REQUEST\_GET\_CURRENT\_CALLS消息，真正的逻辑处理在processSolicited方法中进行来了处理

|  |
| --- |
| private RILRequest processSolicited (Parcel p) {  ...  case RIL\_REQUEST\_GET\_CURRENT\_CALLS: ret = responseCallList(p); break;  ...  if (rr.mResult != null) {  AsyncResult.forMessage(rr.mResult, ret, null);  rr.mResult.sendToTarget();//发出handler消息通知  } |

## 4、EVENT\_POLL\_CALLS\_RESULT消息处理

回到framworks/opt/telephony/…/telephony/gsm/GSMCallTracker.java

rr.mResult.sendToTarget()发出handler消息通知后，会在GSMCallTracker中的handleMessage方法中响应。并且它的消息类型是“EVENT\_POLL\_CALLS\_RESULT”

|  |
| --- |
| @Override  public void handleMessage (Message msg) {  ...  case EVENT\_POLL\_CALLS\_RESULT:  ar = (AsyncResult)msg.obj;  if (msg == mLastRelevantPoll) {  mNeedsPoll = false;  mLastRelevantPoll = null;  handlePollCalls((AsyncResult)msg.obj);  }  break; |

## 5、handlePollCalls方法实现

handlePollCalls方法根据RIL发出的Call List对象判断Call的状态，并发出不同的通知，

1） 新来电的通知是： phone.notifyNewRingingConnection;  
另外两个是  
2) 通话断开通知 onDisconnected；  
3) Call状态变化通知 phone.notifiyPreciseCallStateChanged.  
（当状态改变之后便会通过GsmPhone的notifyPreciseCallStateChanged()方法发起响应）  
来电的时候发出的是phone.notifyNewRingConnection通知，进入到notifyNewRingConnection方法

在handlePollCalls方法中GsmCallTracker通过Handler得到最新的的Call List,遍历mConnections保存通话连接对象列表，通过对应的DriverCall List中对应的DriverCall 对象更新GsmCallTracker中相应的通话相关的信息。其中conn代表旧的通话连接，dc代表新的通话连接。有新的电话进来时，通过dc创建一个新的connection对象进行数据存储.

|  |
| --- |
| @[Override](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=Override&project=odm-foxconn-android-m-kanuti-tulip-somc)  protected synchronized void [handlePollCalls](http://opengrok.sonymobile.net/JDM-BRANCHES/s?refs=handlePollCalls&project=odm-foxconn-android-m-kanuti-tulip-somc)([AsyncResult](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=AsyncResult&project=odm-foxconn-android-m-kanuti-tulip-somc) [ar](http://opengrok.sonymobile.net/JDM-BRANCHES/s?refs=ar&project=odm-foxconn-android-m-kanuti-tulip-somc)) {//处理新来电new Ring  [List](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=List&project=odm-foxconn-android-m-kanuti-tulip-somc) [polledCalls](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=polledCalls&project=odm-foxconn-android-m-kanuti-tulip-somc);// 声明Call List对象，它将保存RIL查询出的当前Call对象列表  if ([ar](http://opengrok.sonymobile.net/JDM-BRANCHES/xref/odm-foxconn-android-m-kanuti-tulip-somc/frameworks/opt/telephony/src/java/com/android/internal/telephony/gsm/GsmCallTracker.java#ar).[exception](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=exception&project=odm-foxconn-android-m-kanuti-tulip-somc) == [null](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=null&project=odm-foxconn-android-m-kanuti-tulip-somc)) {  [polledCalls](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=polledCalls&project=odm-foxconn-android-m-kanuti-tulip-somc) = ([List](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=List&project=odm-foxconn-android-m-kanuti-tulip-somc))[ar](http://opengrok.sonymobile.net/JDM-BRANCHES/xref/odm-foxconn-android-m-kanuti-tulip-somc/frameworks/opt/telephony/src/java/com/android/internal/telephony/gsm/GsmCallTracker.java#ar).[result](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=result&project=odm-foxconn-android-m-kanuti-tulip-somc);//从消息中获取到calllist  } else if ([isCommandExceptionRadioNotAvailable](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=isCommandExceptionRadioNotAvailable&project=odm-foxconn-android-m-kanuti-tulip-somc)([ar](http://opengrok.sonymobile.net/JDM-BRANCHES/xref/odm-foxconn-android-m-kanuti-tulip-somc/frameworks/opt/telephony/src/java/com/android/internal/telephony/gsm/GsmCallTracker.java#ar).[exception](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=exception&project=odm-foxconn-android-m-kanuti-tulip-somc))) {  // just a dummy empty ArrayList to cause the loop  // to hang up all the calls  [polledCalls](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=polledCalls&project=odm-foxconn-android-m-kanuti-tulip-somc) = new [ArrayList](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=ArrayList&project=odm-foxconn-android-m-kanuti-tulip-somc)();  } else {  // Radio probably wasn't ready--try again in a bit  // But don't keep polling if the channel is closed  [pollCallsAfterDelay](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=pollCallsAfterDelay&project=odm-foxconn-android-m-kanuti-tulip-somc)();  return;  }  [Connection](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=Connection&project=odm-foxconn-android-m-kanuti-tulip-somc) [newRinging](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=newRinging&project=odm-foxconn-android-m-kanuti-tulip-somc) = [null](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=null&project=odm-foxconn-android-m-kanuti-tulip-somc); //or waiting //创建通话连接对象Connection对象  [ArrayList](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=ArrayList&project=odm-foxconn-android-m-kanuti-tulip-somc)<[Connection](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=Connection&project=odm-foxconn-android-m-kanuti-tulip-somc)> [newUnknownConnections](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=newUnknownConnections&project=odm-foxconn-android-m-kanuti-tulip-somc) = new [ArrayList](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=ArrayList&project=odm-foxconn-android-m-kanuti-tulip-somc)<[Connection](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=Connection&project=odm-foxconn-android-m-kanuti-tulip-somc)>();  boolean [hasNonHangupStateChanged](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=hasNonHangupStateChanged&project=odm-foxconn-android-m-kanuti-tulip-somc) = false; // Any change besides  // a dropped connection  boolean [hasAnyCallDisconnected](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=hasAnyCallDisconnected&project=odm-foxconn-android-m-kanuti-tulip-somc) = false;  boolean [needsPollDelay](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=needsPollDelay&project=odm-foxconn-android-m-kanuti-tulip-somc) = false;  boolean [unknownConnectionAppeared](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=unknownConnectionAppeared&project=odm-foxconn-android-m-kanuti-tulip-somc) = false;  for (int i = 0, [curDC](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=curDC&project=odm-foxconn-android-m-kanuti-tulip-somc) = 0, [dcSize](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=dcSize&project=odm-foxconn-android-m-kanuti-tulip-somc) = [polledCalls](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=polledCalls&project=odm-foxconn-android-m-kanuti-tulip-somc).[size](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=size&project=odm-foxconn-android-m-kanuti-tulip-somc)()  ; i < [mConnections](http://opengrok.sonymobile.net/JDM-BRANCHES/xref/odm-foxconn-android-m-kanuti-tulip-somc/frameworks/opt/telephony/src/java/com/android/internal/telephony/gsm/GsmCallTracker.java#mConnections).[length](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=length&project=odm-foxconn-android-m-kanuti-tulip-somc); i++) {  [GsmConnection](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=GsmConnection&project=odm-foxconn-android-m-kanuti-tulip-somc) [conn](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=conn&project=odm-foxconn-android-m-kanuti-tulip-somc) = mConnections[i];  [DriverCall](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=DriverCall&project=odm-foxconn-android-m-kanuti-tulip-somc) [dc](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=dc&project=odm-foxconn-android-m-kanuti-tulip-somc) = [null](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=null&project=odm-foxconn-android-m-kanuti-tulip-somc);  // polledCall list is sparse  if ([curDC](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=curDC&project=odm-foxconn-android-m-kanuti-tulip-somc) < [dcSize](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=dcSize&project=odm-foxconn-android-m-kanuti-tulip-somc)) {  [dc](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=dc&project=odm-foxconn-android-m-kanuti-tulip-somc) = ([DriverCall](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=DriverCall&project=odm-foxconn-android-m-kanuti-tulip-somc)) [polledCalls](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=polledCalls&project=odm-foxconn-android-m-kanuti-tulip-somc).[get](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=get&project=odm-foxconn-android-m-kanuti-tulip-somc)([curDC](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=curDC&project=odm-foxconn-android-m-kanuti-tulip-somc)); //得到当前的DriverCall  if ([dc](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=dc&project=odm-foxconn-android-m-kanuti-tulip-somc).[index](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=index&project=odm-foxconn-android-m-kanuti-tulip-somc) == i+1) {  [curDC](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=curDC&project=odm-foxconn-android-m-kanuti-tulip-somc)++;  } else {  [dc](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=dc&project=odm-foxconn-android-m-kanuti-tulip-somc) = [null](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=null&project=odm-foxconn-android-m-kanuti-tulip-somc);  }  }  if ([DBG\_POLL](http://opengrok.sonymobile.net/JDM-BRANCHES/xref/odm-foxconn-android-m-kanuti-tulip-somc/frameworks/opt/telephony/src/java/com/android/internal/telephony/gsm/GsmCallTracker.java#DBG_POLL)) [log](http://opengrok.sonymobile.net/JDM-BRANCHES/xref/odm-foxconn-android-m-kanuti-tulip-somc/frameworks/opt/telephony/src/java/com/android/internal/telephony/gsm/GsmCallTracker.java#log)("poll: conn[i=" + i + "]=" +  [conn](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=conn&project=odm-foxconn-android-m-kanuti-tulip-somc)+", dc=" + [dc](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=dc&project=odm-foxconn-android-m-kanuti-tulip-somc));  if ([conn](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=conn&project=odm-foxconn-android-m-kanuti-tulip-somc) == [null](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=null&project=odm-foxconn-android-m-kanuti-tulip-somc) && [dc](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=dc&project=odm-foxconn-android-m-kanuti-tulip-somc) != [null](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=null&project=odm-foxconn-android-m-kanuti-tulip-somc)) {  // Connection appeared in CLCC response that we don't know about  if ([mPendingMO](http://opengrok.sonymobile.net/JDM-BRANCHES/xref/odm-foxconn-android-m-kanuti-tulip-somc/frameworks/opt/telephony/src/java/com/android/internal/telephony/gsm/GsmCallTracker.java#mPendingMO) != [null](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=null&project=odm-foxconn-android-m-kanuti-tulip-somc) && [mPendingMO](http://opengrok.sonymobile.net/JDM-BRANCHES/xref/odm-foxconn-android-m-kanuti-tulip-somc/frameworks/opt/telephony/src/java/com/android/internal/telephony/gsm/GsmCallTracker.java#mPendingMO).[compareTo](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=compareTo&project=odm-foxconn-android-m-kanuti-tulip-somc)([dc](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=dc&project=odm-foxconn-android-m-kanuti-tulip-somc))) {  if ([DBG\_POLL](http://opengrok.sonymobile.net/JDM-BRANCHES/xref/odm-foxconn-android-m-kanuti-tulip-somc/frameworks/opt/telephony/src/java/com/android/internal/telephony/gsm/GsmCallTracker.java#DBG_POLL)) [log](http://opengrok.sonymobile.net/JDM-BRANCHES/xref/odm-foxconn-android-m-kanuti-tulip-somc/frameworks/opt/telephony/src/java/com/android/internal/telephony/gsm/GsmCallTracker.java#log)("poll: pendingMO=" + [mPendingMO](http://opengrok.sonymobile.net/JDM-BRANCHES/xref/odm-foxconn-android-m-kanuti-tulip-somc/frameworks/opt/telephony/src/java/com/android/internal/telephony/gsm/GsmCallTracker.java#mPendingMO));  // It's our pending mobile originating call  [mConnections](http://opengrok.sonymobile.net/JDM-BRANCHES/xref/odm-foxconn-android-m-kanuti-tulip-somc/frameworks/opt/telephony/src/java/com/android/internal/telephony/gsm/GsmCallTracker.java#mConnections)[i] = [mPendingMO](http://opengrok.sonymobile.net/JDM-BRANCHES/xref/odm-foxconn-android-m-kanuti-tulip-somc/frameworks/opt/telephony/src/java/com/android/internal/telephony/gsm/GsmCallTracker.java#mPendingMO);  [mPendingMO](http://opengrok.sonymobile.net/JDM-BRANCHES/xref/odm-foxconn-android-m-kanuti-tulip-somc/frameworks/opt/telephony/src/java/com/android/internal/telephony/gsm/GsmCallTracker.java#mPendingMO).[mIndex](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=mIndex&project=odm-foxconn-android-m-kanuti-tulip-somc) = i;  [mPendingMO](http://opengrok.sonymobile.net/JDM-BRANCHES/xref/odm-foxconn-android-m-kanuti-tulip-somc/frameworks/opt/telephony/src/java/com/android/internal/telephony/gsm/GsmCallTracker.java#mPendingMO).[update](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=update&project=odm-foxconn-android-m-kanuti-tulip-somc)([dc](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=dc&project=odm-foxconn-android-m-kanuti-tulip-somc));  [mPendingMO](http://opengrok.sonymobile.net/JDM-BRANCHES/xref/odm-foxconn-android-m-kanuti-tulip-somc/frameworks/opt/telephony/src/java/com/android/internal/telephony/gsm/GsmCallTracker.java#mPendingMO) = [null](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=null&project=odm-foxconn-android-m-kanuti-tulip-somc);  // Someone has already asked to hangup this call  if ([mHangupPendingMO](http://opengrok.sonymobile.net/JDM-BRANCHES/xref/odm-foxconn-android-m-kanuti-tulip-somc/frameworks/opt/telephony/src/java/com/android/internal/telephony/gsm/GsmCallTracker.java#mHangupPendingMO)) {  [mHangupPendingMO](http://opengrok.sonymobile.net/JDM-BRANCHES/xref/odm-foxconn-android-m-kanuti-tulip-somc/frameworks/opt/telephony/src/java/com/android/internal/telephony/gsm/GsmCallTracker.java#mHangupPendingMO) = false;  try {  if ([Phone](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=Phone&project=odm-foxconn-android-m-kanuti-tulip-somc).[DEBUG\_PHONE](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=DEBUG_PHONE&project=odm-foxconn-android-m-kanuti-tulip-somc)) [log](http://opengrok.sonymobile.net/JDM-BRANCHES/xref/odm-foxconn-android-m-kanuti-tulip-somc/frameworks/opt/telephony/src/java/com/android/internal/telephony/gsm/GsmCallTracker.java#log)(  "poll: hangupPendingMO, hangup conn " + i);  [hangup](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=hangup&project=odm-foxconn-android-m-kanuti-tulip-somc)([mConnections](http://opengrok.sonymobile.net/JDM-BRANCHES/xref/odm-foxconn-android-m-kanuti-tulip-somc/frameworks/opt/telephony/src/java/com/android/internal/telephony/gsm/GsmCallTracker.java#mConnections)[i]);  } catch ([CallStateException](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=CallStateException&project=odm-foxconn-android-m-kanuti-tulip-somc) [ex](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=ex&project=odm-foxconn-android-m-kanuti-tulip-somc)) {  [Rlog](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=Rlog&project=odm-foxconn-android-m-kanuti-tulip-somc).e([LOG\_TAG](http://opengrok.sonymobile.net/JDM-BRANCHES/xref/odm-foxconn-android-m-kanuti-tulip-somc/frameworks/opt/telephony/src/java/com/android/internal/telephony/gsm/GsmCallTracker.java#LOG_TAG), "unexpected error on hangup");  }  // Do not continue processing this poll  // Wait for hangup and repoll  return;  }  } else {  [mConnections](http://opengrok.sonymobile.net/JDM-BRANCHES/xref/odm-foxconn-android-m-kanuti-tulip-somc/frameworks/opt/telephony/src/java/com/android/internal/telephony/gsm/GsmCallTracker.java#mConnections)[i] = new [GsmConnection](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=GsmConnection&project=odm-foxconn-android-m-kanuti-tulip-somc)([mPhone](http://opengrok.sonymobile.net/JDM-BRANCHES/xref/odm-foxconn-android-m-kanuti-tulip-somc/frameworks/opt/telephony/src/java/com/android/internal/telephony/gsm/GsmCallTracker.java#mPhone), [dc](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=dc&project=odm-foxconn-android-m-kanuti-tulip-somc), this, i);//当前无连接时创建  [Connection](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=Connection&project=odm-foxconn-android-m-kanuti-tulip-somc) [hoConnection](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=hoConnection&project=odm-foxconn-android-m-kanuti-tulip-somc) = [getHoConnection](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=getHoConnection&project=odm-foxconn-android-m-kanuti-tulip-somc)([dc](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=dc&project=odm-foxconn-android-m-kanuti-tulip-somc));  if ([hoConnection](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=hoConnection&project=odm-foxconn-android-m-kanuti-tulip-somc) != [null](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=null&project=odm-foxconn-android-m-kanuti-tulip-somc)) // Single Radio Voice Call Continuity (SRVCC) completed  [mConnections](http://opengrok.sonymobile.net/JDM-BRANCHES/xref/odm-foxconn-android-m-kanuti-tulip-somc/frameworks/opt/telephony/src/java/com/android/internal/telephony/gsm/GsmCallTracker.java#mConnections)[i].[migrateFrom](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=migrateFrom&project=odm-foxconn-android-m-kanuti-tulip-somc)([hoConnection](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=hoConnection&project=odm-foxconn-android-m-kanuti-tulip-somc));  // Updating connect time for silent redial cases (ex: Calls are transferred  // from [DIALING](http://opengrok.sonymobile.net/JDM-BRANCHES/s?path=DIALING/)/[ALERTING](http://opengrok.sonymobile.net/JDM-BRANCHES/s?path=DIALING/ALERTING/)/[INCOMING](http://opengrok.sonymobile.net/JDM-BRANCHES/s?path=DIALING/ALERTING/INCOMING/)/[WAITING](http://opengrok.sonymobile.net/JDM-BRANCHES/s?path=DIALING/ALERTING/INCOMING/WAITING) to ACTIVE)  if ([hoConnection](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=hoConnection&project=odm-foxconn-android-m-kanuti-tulip-somc).[mPreHandoverState](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=mPreHandoverState&project=odm-foxconn-android-m-kanuti-tulip-somc) != [GsmCall](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=GsmCall&project=odm-foxconn-android-m-kanuti-tulip-somc).[State](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=State&project=odm-foxconn-android-m-kanuti-tulip-somc).[ACTIVE](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=ACTIVE&project=odm-foxconn-android-m-kanuti-tulip-somc) &&  [hoConnection](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=hoConnection&project=odm-foxconn-android-m-kanuti-tulip-somc).[mPreHandoverState](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=mPreHandoverState&project=odm-foxconn-android-m-kanuti-tulip-somc) != [GsmCall](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=GsmCall&project=odm-foxconn-android-m-kanuti-tulip-somc).[State](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=State&project=odm-foxconn-android-m-kanuti-tulip-somc).[HOLDING](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=HOLDING&project=odm-foxconn-android-m-kanuti-tulip-somc)) {  [mConnections](http://opengrok.sonymobile.net/JDM-BRANCHES/xref/odm-foxconn-android-m-kanuti-tulip-somc/frameworks/opt/telephony/src/java/com/android/internal/telephony/gsm/GsmCallTracker.java#mConnections)[i].[onConnectedInOrOut](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=onConnectedInOrOut&project=odm-foxconn-android-m-kanuti-tulip-somc)();  }  [mHandoverConnections](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=mHandoverConnections&project=odm-foxconn-android-m-kanuti-tulip-somc).[remove](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=remove&project=odm-foxconn-android-m-kanuti-tulip-somc)([hoConnection](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=hoConnection&project=odm-foxconn-android-m-kanuti-tulip-somc));  for ([Iterator](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=Iterator&project=odm-foxconn-android-m-kanuti-tulip-somc)<[Connection](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=Connection&project=odm-foxconn-android-m-kanuti-tulip-somc)> [it](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=it&project=odm-foxconn-android-m-kanuti-tulip-somc) = [mHandoverConnections](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=mHandoverConnections&project=odm-foxconn-android-m-kanuti-tulip-somc).[iterator](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=iterator&project=odm-foxconn-android-m-kanuti-tulip-somc)();  [it](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=it&project=odm-foxconn-android-m-kanuti-tulip-somc).[hasNext](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=hasNext&project=odm-foxconn-android-m-kanuti-tulip-somc)();) {  [Connection](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=Connection&project=odm-foxconn-android-m-kanuti-tulip-somc) c = [it](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=it&project=odm-foxconn-android-m-kanuti-tulip-somc).[next](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=next&project=odm-foxconn-android-m-kanuti-tulip-somc)();  [Rlog](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=Rlog&project=odm-foxconn-android-m-kanuti-tulip-somc).i([LOG\_TAG](http://opengrok.sonymobile.net/JDM-BRANCHES/xref/odm-foxconn-android-m-kanuti-tulip-somc/frameworks/opt/telephony/src/java/com/android/internal/telephony/gsm/GsmCallTracker.java#LOG_TAG), "HO Conn state is " + c.[mPreHandoverState](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=mPreHandoverState&project=odm-foxconn-android-m-kanuti-tulip-somc));  if (c.[mPreHandoverState](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=mPreHandoverState&project=odm-foxconn-android-m-kanuti-tulip-somc) == [mConnections](http://opengrok.sonymobile.net/JDM-BRANCHES/xref/odm-foxconn-android-m-kanuti-tulip-somc/frameworks/opt/telephony/src/java/com/android/internal/telephony/gsm/GsmCallTracker.java#mConnections)[i].[getState](http://opengrok.sonymobile.net/JDM-BRANCHES/xref/odm-foxconn-android-m-kanuti-tulip-somc/frameworks/opt/telephony/src/java/com/android/internal/telephony/gsm/GsmCallTracker.java#getState)()) {  [Rlog](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=Rlog&project=odm-foxconn-android-m-kanuti-tulip-somc).i([LOG\_TAG](http://opengrok.sonymobile.net/JDM-BRANCHES/xref/odm-foxconn-android-m-kanuti-tulip-somc/frameworks/opt/telephony/src/java/com/android/internal/telephony/gsm/GsmCallTracker.java#LOG_TAG), "Removing HO conn "  + [hoConnection](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=hoConnection&project=odm-foxconn-android-m-kanuti-tulip-somc) + c.[mPreHandoverState](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=mPreHandoverState&project=odm-foxconn-android-m-kanuti-tulip-somc));  [it](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=it&project=odm-foxconn-android-m-kanuti-tulip-somc).[remove](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=remove&project=odm-foxconn-android-m-kanuti-tulip-somc)();  }  }  [mPhone](http://opengrok.sonymobile.net/JDM-BRANCHES/xref/odm-foxconn-android-m-kanuti-tulip-somc/frameworks/opt/telephony/src/java/com/android/internal/telephony/gsm/GsmCallTracker.java#mPhone).[notifyHandoverStateChanged](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=notifyHandoverStateChanged&project=odm-foxconn-android-m-kanuti-tulip-somc)([mConnections](http://opengrok.sonymobile.net/JDM-BRANCHES/xref/odm-foxconn-android-m-kanuti-tulip-somc/frameworks/opt/telephony/src/java/com/android/internal/telephony/gsm/GsmCallTracker.java#mConnections)[i]);  } else if ( [mConnections](http://opengrok.sonymobile.net/JDM-BRANCHES/xref/odm-foxconn-android-m-kanuti-tulip-somc/frameworks/opt/telephony/src/java/com/android/internal/telephony/gsm/GsmCallTracker.java#mConnections)[i].[getCall](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=getCall&project=odm-foxconn-android-m-kanuti-tulip-somc)() == [mRingingCall](http://opengrok.sonymobile.net/JDM-BRANCHES/xref/odm-foxconn-android-m-kanuti-tulip-somc/frameworks/opt/telephony/src/java/com/android/internal/telephony/gsm/GsmCallTracker.java#mRingingCall) ) { // it's a ringing call  [newRinging](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=newRinging&project=odm-foxconn-android-m-kanuti-tulip-somc) = [mConnections](http://opengrok.sonymobile.net/JDM-BRANCHES/xref/odm-foxconn-android-m-kanuti-tulip-somc/frameworks/opt/telephony/src/java/com/android/internal/telephony/gsm/GsmCallTracker.java#mConnections)[i];//当有电话进来时  } else {  if ([dc](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=dc&project=odm-foxconn-android-m-kanuti-tulip-somc).[state](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=state&project=odm-foxconn-android-m-kanuti-tulip-somc) != [DriverCall](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=DriverCall&project=odm-foxconn-android-m-kanuti-tulip-somc).[State](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=State&project=odm-foxconn-android-m-kanuti-tulip-somc).[ALERTING](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=ALERTING&project=odm-foxconn-android-m-kanuti-tulip-somc)  && [dc](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=dc&project=odm-foxconn-android-m-kanuti-tulip-somc).[state](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=state&project=odm-foxconn-android-m-kanuti-tulip-somc) != [DriverCall](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=DriverCall&project=odm-foxconn-android-m-kanuti-tulip-somc).[State](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=State&project=odm-foxconn-android-m-kanuti-tulip-somc).[DIALING](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=DIALING&project=odm-foxconn-android-m-kanuti-tulip-somc)) {  [mConnections](http://opengrok.sonymobile.net/JDM-BRANCHES/xref/odm-foxconn-android-m-kanuti-tulip-somc/frameworks/opt/telephony/src/java/com/android/internal/telephony/gsm/GsmCallTracker.java#mConnections)[i].[onConnectedInOrOut](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=onConnectedInOrOut&project=odm-foxconn-android-m-kanuti-tulip-somc)();  if ([dc](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=dc&project=odm-foxconn-android-m-kanuti-tulip-somc).[state](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=state&project=odm-foxconn-android-m-kanuti-tulip-somc) == [DriverCall](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=DriverCall&project=odm-foxconn-android-m-kanuti-tulip-somc).[State](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=State&project=odm-foxconn-android-m-kanuti-tulip-somc).[HOLDING](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=HOLDING&project=odm-foxconn-android-m-kanuti-tulip-somc)) {  // We've transitioned into HOLDING  [mConnections](http://opengrok.sonymobile.net/JDM-BRANCHES/xref/odm-foxconn-android-m-kanuti-tulip-somc/frameworks/opt/telephony/src/java/com/android/internal/telephony/gsm/GsmCallTracker.java#mConnections)[i].[onStartedHolding](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=onStartedHolding&project=odm-foxconn-android-m-kanuti-tulip-somc)();  }  }  [newUnknownConnections](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=newUnknownConnections&project=odm-foxconn-android-m-kanuti-tulip-somc).[add](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=add&project=odm-foxconn-android-m-kanuti-tulip-somc)([mConnections](http://opengrok.sonymobile.net/JDM-BRANCHES/xref/odm-foxconn-android-m-kanuti-tulip-somc/frameworks/opt/telephony/src/java/com/android/internal/telephony/gsm/GsmCallTracker.java#mConnections)[i]);  [unknownConnectionAppeared](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=unknownConnectionAppeared&project=odm-foxconn-android-m-kanuti-tulip-somc) = true;  }  }  [hasNonHangupStateChanged](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=hasNonHangupStateChanged&project=odm-foxconn-android-m-kanuti-tulip-somc) = true;  } else if ([conn](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=conn&project=odm-foxconn-android-m-kanuti-tulip-somc) != [null](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=null&project=odm-foxconn-android-m-kanuti-tulip-somc) && [dc](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=dc&project=odm-foxconn-android-m-kanuti-tulip-somc) == [null](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=null&project=odm-foxconn-android-m-kanuti-tulip-somc)) {//通话连接断开  [log](http://opengrok.sonymobile.net/JDM-BRANCHES/xref/odm-foxconn-android-m-kanuti-tulip-somc/frameworks/opt/telephony/src/java/com/android/internal/telephony/gsm/GsmCallTracker.java#log)("[SW12]conn is not null, dc is null.");//SW12-WC-DEBUG\_LOG-01+  [mDroppedDuringPoll](http://opengrok.sonymobile.net/JDM-BRANCHES/xref/odm-foxconn-android-m-kanuti-tulip-somc/frameworks/opt/telephony/src/java/com/android/internal/telephony/gsm/GsmCallTracker.java#mDroppedDuringPoll).[add](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=add&project=odm-foxconn-android-m-kanuti-tulip-somc)([conn](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=conn&project=odm-foxconn-android-m-kanuti-tulip-somc));  [mConnections](http://opengrok.sonymobile.net/JDM-BRANCHES/xref/odm-foxconn-android-m-kanuti-tulip-somc/frameworks/opt/telephony/src/java/com/android/internal/telephony/gsm/GsmCallTracker.java#mConnections)[i] = [null](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=null&project=odm-foxconn-android-m-kanuti-tulip-somc);  } else if ([conn](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=conn&project=odm-foxconn-android-m-kanuti-tulip-somc) != [null](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=null&project=odm-foxconn-android-m-kanuti-tulip-somc) && [dc](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=dc&project=odm-foxconn-android-m-kanuti-tulip-somc) != [null](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=null&project=odm-foxconn-android-m-kanuti-tulip-somc) && ![conn](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=conn&project=odm-foxconn-android-m-kanuti-tulip-somc).[compareTo](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=compareTo&project=odm-foxconn-android-m-kanuti-tulip-somc)([dc](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=dc&project=odm-foxconn-android-m-kanuti-tulip-somc))) {  [log](http://opengrok.sonymobile.net/JDM-BRANCHES/xref/odm-foxconn-android-m-kanuti-tulip-somc/frameworks/opt/telephony/src/java/com/android/internal/telephony/gsm/GsmCallTracker.java#log)("[SW12]Connection in CLCC response does not match");//SW12-WC-DEBUG\_LOG-01+  [mDroppedDuringPoll](http://opengrok.sonymobile.net/JDM-BRANCHES/xref/odm-foxconn-android-m-kanuti-tulip-somc/frameworks/opt/telephony/src/java/com/android/internal/telephony/gsm/GsmCallTracker.java#mDroppedDuringPoll).[add](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=add&project=odm-foxconn-android-m-kanuti-tulip-somc)([conn](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=conn&project=odm-foxconn-android-m-kanuti-tulip-somc));  [mConnections](http://opengrok.sonymobile.net/JDM-BRANCHES/xref/odm-foxconn-android-m-kanuti-tulip-somc/frameworks/opt/telephony/src/java/com/android/internal/telephony/gsm/GsmCallTracker.java#mConnections)[i] = new [GsmConnection](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=GsmConnection&project=odm-foxconn-android-m-kanuti-tulip-somc) ([mPhone](http://opengrok.sonymobile.net/JDM-BRANCHES/xref/odm-foxconn-android-m-kanuti-tulip-somc/frameworks/opt/telephony/src/java/com/android/internal/telephony/gsm/GsmCallTracker.java#mPhone), [dc](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=dc&project=odm-foxconn-android-m-kanuti-tulip-somc), this, i);  if ([mConnections](http://opengrok.sonymobile.net/JDM-BRANCHES/xref/odm-foxconn-android-m-kanuti-tulip-somc/frameworks/opt/telephony/src/java/com/android/internal/telephony/gsm/GsmCallTracker.java#mConnections)[i].[getCall](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=getCall&project=odm-foxconn-android-m-kanuti-tulip-somc)() == [mRingingCall](http://opengrok.sonymobile.net/JDM-BRANCHES/xref/odm-foxconn-android-m-kanuti-tulip-somc/frameworks/opt/telephony/src/java/com/android/internal/telephony/gsm/GsmCallTracker.java#mRingingCall)) {  [newRinging](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=newRinging&project=odm-foxconn-android-m-kanuti-tulip-somc) = [mConnections](http://opengrok.sonymobile.net/JDM-BRANCHES/xref/odm-foxconn-android-m-kanuti-tulip-somc/frameworks/opt/telephony/src/java/com/android/internal/telephony/gsm/GsmCallTracker.java#mConnections)[i];  } // else something strange happened  [hasNonHangupStateChanged](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=hasNonHangupStateChanged&project=odm-foxconn-android-m-kanuti-tulip-somc) = true;  } else if ([conn](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=conn&project=odm-foxconn-android-m-kanuti-tulip-somc) != [null](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=null&project=odm-foxconn-android-m-kanuti-tulip-somc) && [dc](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=dc&project=odm-foxconn-android-m-kanuti-tulip-somc) != [null](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=null&project=odm-foxconn-android-m-kanuti-tulip-somc)) { /\* implicit conn.compareTo(dc) \*/通话状态发生变化  [log](http://opengrok.sonymobile.net/JDM-BRANCHES/xref/odm-foxconn-android-m-kanuti-tulip-somc/frameworks/opt/telephony/src/java/com/android/internal/telephony/gsm/GsmCallTracker.java#log)("[SW12]conn is not null, and dc is not null.");//SW12-WC-DEBUG\_LOG-01+  boolean [changed](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=changed&project=odm-foxconn-android-m-kanuti-tulip-somc);  [changed](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=changed&project=odm-foxconn-android-m-kanuti-tulip-somc) = [conn](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=conn&project=odm-foxconn-android-m-kanuti-tulip-somc).[update](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=update&project=odm-foxconn-android-m-kanuti-tulip-somc)([dc](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=dc&project=odm-foxconn-android-m-kanuti-tulip-somc));//通话连接发生变化时，通过dc更新conn  [hasNonHangupStateChanged](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=hasNonHangupStateChanged&project=odm-foxconn-android-m-kanuti-tulip-somc) = [hasNonHangupStateChanged](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=hasNonHangupStateChanged&project=odm-foxconn-android-m-kanuti-tulip-somc) || [changed](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=changed&project=odm-foxconn-android-m-kanuti-tulip-somc);  }  if ([REPEAT\_POLLING](http://opengrok.sonymobile.net/JDM-BRANCHES/xref/odm-foxconn-android-m-kanuti-tulip-somc/frameworks/opt/telephony/src/java/com/android/internal/telephony/gsm/GsmCallTracker.java#REPEAT_POLLING)) {  if ([dc](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=dc&project=odm-foxconn-android-m-kanuti-tulip-somc) != [null](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=null&project=odm-foxconn-android-m-kanuti-tulip-somc)) {  // FIXME with RIL, we should not need this anymore  if (([dc](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=dc&project=odm-foxconn-android-m-kanuti-tulip-somc).[state](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=state&project=odm-foxconn-android-m-kanuti-tulip-somc) == [DriverCall](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=DriverCall&project=odm-foxconn-android-m-kanuti-tulip-somc).[State](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=State&project=odm-foxconn-android-m-kanuti-tulip-somc).[DIALING](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=DIALING&project=odm-foxconn-android-m-kanuti-tulip-somc)  /\*&& cm.getOption(cm.OPTION\_POLL\_DIALING)\*/)  || ([dc](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=dc&project=odm-foxconn-android-m-kanuti-tulip-somc).[state](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=state&project=odm-foxconn-android-m-kanuti-tulip-somc) == [DriverCall](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=DriverCall&project=odm-foxconn-android-m-kanuti-tulip-somc).[State](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=State&project=odm-foxconn-android-m-kanuti-tulip-somc).[ALERTING](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=ALERTING&project=odm-foxconn-android-m-kanuti-tulip-somc)  /\*&& cm.getOption(cm.OPTION\_POLL\_ALERTING)\*/)  || ([dc](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=dc&project=odm-foxconn-android-m-kanuti-tulip-somc).[state](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=state&project=odm-foxconn-android-m-kanuti-tulip-somc) == [DriverCall](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=DriverCall&project=odm-foxconn-android-m-kanuti-tulip-somc).[State](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=State&project=odm-foxconn-android-m-kanuti-tulip-somc).[INCOMING](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=INCOMING&project=odm-foxconn-android-m-kanuti-tulip-somc)  /\*&& cm.getOption(cm.OPTION\_POLL\_INCOMING)\*/)  || ([dc](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=dc&project=odm-foxconn-android-m-kanuti-tulip-somc).[state](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=state&project=odm-foxconn-android-m-kanuti-tulip-somc) == [DriverCall](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=DriverCall&project=odm-foxconn-android-m-kanuti-tulip-somc).[State](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=State&project=odm-foxconn-android-m-kanuti-tulip-somc).[WAITING](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=WAITING&project=odm-foxconn-android-m-kanuti-tulip-somc)  /\*&& cm.getOption(cm.OPTION\_POLL\_WAITING)\*/)  ) {  [needsPollDelay](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=needsPollDelay&project=odm-foxconn-android-m-kanuti-tulip-somc) = true;  }  }  }  }  if ([mPendingMO](http://opengrok.sonymobile.net/JDM-BRANCHES/xref/odm-foxconn-android-m-kanuti-tulip-somc/frameworks/opt/telephony/src/java/com/android/internal/telephony/gsm/GsmCallTracker.java#mPendingMO) != [null](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=null&project=odm-foxconn-android-m-kanuti-tulip-somc)) {  [Rlog](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=Rlog&project=odm-foxconn-android-m-kanuti-tulip-somc).d([LOG\_TAG](http://opengrok.sonymobile.net/JDM-BRANCHES/xref/odm-foxconn-android-m-kanuti-tulip-somc/frameworks/opt/telephony/src/java/com/android/internal/telephony/gsm/GsmCallTracker.java#LOG_TAG),"Pending MO dropped before poll fg state:"  + [mForegroundCall](http://opengrok.sonymobile.net/JDM-BRANCHES/xref/odm-foxconn-android-m-kanuti-tulip-somc/frameworks/opt/telephony/src/java/com/android/internal/telephony/gsm/GsmCallTracker.java#mForegroundCall).[getState](http://opengrok.sonymobile.net/JDM-BRANCHES/xref/odm-foxconn-android-m-kanuti-tulip-somc/frameworks/opt/telephony/src/java/com/android/internal/telephony/gsm/GsmCallTracker.java#getState)());  [mDroppedDuringPoll](http://opengrok.sonymobile.net/JDM-BRANCHES/xref/odm-foxconn-android-m-kanuti-tulip-somc/frameworks/opt/telephony/src/java/com/android/internal/telephony/gsm/GsmCallTracker.java#mDroppedDuringPoll).[add](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=add&project=odm-foxconn-android-m-kanuti-tulip-somc)([mPendingMO](http://opengrok.sonymobile.net/JDM-BRANCHES/xref/odm-foxconn-android-m-kanuti-tulip-somc/frameworks/opt/telephony/src/java/com/android/internal/telephony/gsm/GsmCallTracker.java#mPendingMO));  [mPendingMO](http://opengrok.sonymobile.net/JDM-BRANCHES/xref/odm-foxconn-android-m-kanuti-tulip-somc/frameworks/opt/telephony/src/java/com/android/internal/telephony/gsm/GsmCallTracker.java#mPendingMO) = [null](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=null&project=odm-foxconn-android-m-kanuti-tulip-somc);  [mHangupPendingMO](http://opengrok.sonymobile.net/JDM-BRANCHES/xref/odm-foxconn-android-m-kanuti-tulip-somc/frameworks/opt/telephony/src/java/com/android/internal/telephony/gsm/GsmCallTracker.java#mHangupPendingMO) = false;  }  if ([newRinging](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=newRinging&project=odm-foxconn-android-m-kanuti-tulip-somc) != [null](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=null&project=odm-foxconn-android-m-kanuti-tulip-somc)) {//接收到来电请求  [mPhone](http://opengrok.sonymobile.net/JDM-BRANCHES/xref/odm-foxconn-android-m-kanuti-tulip-somc/frameworks/opt/telephony/src/java/com/android/internal/telephony/gsm/GsmCallTracker.java#mPhone).[notifyNewRingingConnection](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=notifyNewRingingConnection&project=odm-foxconn-android-m-kanuti-tulip-somc)([newRinging](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=newRinging&project=odm-foxconn-android-m-kanuti-tulip-somc));  }  ……..  if ([newRinging](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=newRinging&project=odm-foxconn-android-m-kanuti-tulip-somc) != [null](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=null&project=odm-foxconn-android-m-kanuti-tulip-somc) || [hasNonHangupStateChanged](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=hasNonHangupStateChanged&project=odm-foxconn-android-m-kanuti-tulip-somc) || [hasAnyCallDisconnected](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=hasAnyCallDisconnected&project=odm-foxconn-android-m-kanuti-tulip-somc)) {  [internalClearDisconnected](http://opengrok.sonymobile.net/JDM-BRANCHES/xref/odm-foxconn-android-m-kanuti-tulip-somc/frameworks/opt/telephony/src/java/com/android/internal/telephony/gsm/GsmCallTracker.java#internalClearDisconnected)();  }  [updatePhoneState](http://opengrok.sonymobile.net/JDM-BRANCHES/xref/odm-foxconn-android-m-kanuti-tulip-somc/frameworks/opt/telephony/src/java/com/android/internal/telephony/gsm/GsmCallTracker.java" \l "updatePhoneState)();//同步更新Phone.State  if ([unknownConnectionAppeared](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=unknownConnectionAppeared&project=odm-foxconn-android-m-kanuti-tulip-somc)) {//出现未知的通话连接  for ([Connection](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=Connection&project=odm-foxconn-android-m-kanuti-tulip-somc) c : [newUnknownConnections](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=newUnknownConnections&project=odm-foxconn-android-m-kanuti-tulip-somc)) {  [log](http://opengrok.sonymobile.net/JDM-BRANCHES/xref/odm-foxconn-android-m-kanuti-tulip-somc/frameworks/opt/telephony/src/java/com/android/internal/telephony/gsm/GsmCallTracker.java#log)("Notify unknown for " + c);  [mPhone](http://opengrok.sonymobile.net/JDM-BRANCHES/xref/odm-foxconn-android-m-kanuti-tulip-somc/frameworks/opt/telephony/src/java/com/android/internal/telephony/gsm/GsmCallTracker.java#mPhone).[notifyUnknownConnection](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=notifyUnknownConnection&project=odm-foxconn-android-m-kanuti-tulip-somc)(c);  }  }  if ([hasNonHangupStateChanged](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=hasNonHangupStateChanged&project=odm-foxconn-android-m-kanuti-tulip-somc) || [newRinging](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=newRinging&project=odm-foxconn-android-m-kanuti-tulip-somc) != [null](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=null&project=odm-foxconn-android-m-kanuti-tulip-somc) || [hasAnyCallDisconnected](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=hasAnyCallDisconnected&project=odm-foxconn-android-m-kanuti-tulip-somc)) {  [mPhone](http://opengrok.sonymobile.net/JDM-BRANCHES/xref/odm-foxconn-android-m-kanuti-tulip-somc/frameworks/opt/telephony/src/java/com/android/internal/telephony/gsm/GsmCallTracker.java#mPhone).[notifyPreciseCallStateChanged](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=notifyPreciseCallStateChanged&project=odm-foxconn-android-m-kanuti-tulip-somc)();  }  [dumpState](http://opengrok.sonymobile.net/JDM-BRANCHES/xref/odm-foxconn-android-m-kanuti-tulip-somc/frameworks/opt/telephony/src/java/com/android/internal/telephony/gsm/GsmCallTracker.java#dumpState)();  } |

通过dc对象更新conn对象，conn对象调用update方法返回通话状态是否已经更新的标志

|  |
| --- |
| update (DriverCall dc) {  GsmCall newParent;  boolean changed = false;  boolean wasConnectingInOrOut = isConnectingInOrOut();  boolean wasHolding = (getState() == GsmCall.State.HOLDING);  newParent = parentFromDCState(dc.state);//得到当前的drivercall的状态  //Ignore dc.number and dc.name in case of a handover connection  if (mOrigConnection != null) {  if (Phone.DEBUG\_PHONE) log("update: mOrigConnection is not null");  } else {  log(" mNumberConverted " + mNumberConverted);//false  if (!equalsHandlesNulls(mAddress, dc.number) && (!mNumberConverted  || !equalsHandlesNulls(mConvertedNumber, dc.number))) {  if (Phone.DEBUG\_PHONE) log("update: phone # changed!");  mAddress = dc.number;  changed = true;  }  //+{SW12-JC-HUTCH\_HD\_VOICE\_INDICATOR-01  if (!equalsHandlesNulls(mCodec, dc.codec)) {  if (Phone.DEBUG\_PHONE) log("update: codec # changed!");  log("[WP1]Old Codec -> " + mCodec);  log("[WP1]New codec -> " + dc.codec);  mCodec = dc.codec;  changed = true;  }  //SW12-JC-HUTCH\_HD\_VOICE\_INDICATOR-01}+  }  // A null cnapName should be the same as ""  if (TextUtils.isEmpty(dc.name)) {  if (!TextUtils.isEmpty(mCnapName)) {  changed = true;  mCnapName = "";  }  } else if (!dc.name.equals(mCnapName)) {  changed = true;  mCnapName = dc.name;  }  if (Phone.DEBUG\_PHONE) log("--dssds----"+mCnapName);//--dssds----null  mCnapNamePresentation = dc.namePresentation;  mNumberPresentation = dc.numberPresentation;  if (newParent != mParent) {  if (mParent != null) {  mParent.detach(this);  }  newParent.attach(this, dc);  mParent = newParent;  changed = true;  } else {  boolean parentStateChange;  parentStateChange = mParent.update (this, dc);  changed = changed || parentStateChange;  }  /\*\* Some state-transition events \*/  if (Phone.DEBUG\_PHONE) log(  "update: parent=" + mParent +  ", hasNewParent=" + (newParent != mParent) +  ", wasConnectingInOrOut=" + wasConnectingInOrOut +  ", wasHolding=" + wasHolding +  ", isConnectingInOrOut=" + isConnectingInOrOut() +  ", changed=" + changed);  //update: parent=ACTIVE, hasNewParent=false, wasConnectingInOrOut=true, wasHolding=false, isConnectingInOrOut=false, changed=true  if (wasConnectingInOrOut && !isConnectingInOrOut()) {  onConnectedInOrOut();  }  if (changed && !wasHolding && (getState() == GsmCall.State.HOLDING)) {  // We've transitioned into HOLDING  onStartedHolding();  }  return changed;  } |

在UpdatePhoneState方法中首先会记录原先的状态，在根据三个mState状态，更新Phone.State手机状态

|  |
| --- |
| updatePhoneState() {  PhoneConstants.State oldState = mState;  if (mRingingCall.isRinging()) {//来电状态  mState = PhoneConstants.State.RINGING;  } else if (mPendingMO != null ||  !(mForegroundCall.isIdle() && mBackgroundCall.isIdle())) {//  mState = PhoneConstants.State.OFFHOOK;//摘机状态  } else {  ImsPhone imsPhone = (ImsPhone)mPhone.getImsPhone();  if ( mState == PhoneConstants.State.OFFHOOK && (imsPhone != null)){  imsPhone.callEndCleanupHandOverCallIfAny();  }  mState = PhoneConstants.State.IDLE;//待机状态  }  //发送状态通知  if (mState == PhoneConstants.State.IDLE && oldState != mState) {  mVoiceCallEndedRegistrants.notifyRegistrants(  new AsyncResult(null, null, null));  } else if (oldState == PhoneConstants.State.IDLE && oldState != mState) {  mVoiceCallStartedRegistrants.notifyRegistrants (  new AsyncResult(null, null, null));  } |

## 6、notifyRegistrants实现及相应的观察者Registrant

framworks/opt/telephony/…/telephony/gsm/GSMPhone.java

实现notifyNewRingingConnection方法，并调用父类PhoneBase.java（为com.android.internal.telephony.phone 接口实现。）notifyNewRingingConnectionP()发出来电通知mNewRingingConnectionRegistrants.notifyRegistrants (ar); 查看mNewRingingConnectionRegistrants对象是在哪儿定义并实现的，查找相应的Handler进行消息处理

|  |
| --- |
| **RegistrantList.java \frameworks\base\core\java\android\os**\ **notifyRegistrants方法实现**  public /\*synchronized\*/ void notifyRegistrants(AsyncResult ar){  internalNotifyRegistrants(ar.result, ar.exception);  }  private synchronized void internalNotifyRegistrants (Object result, Throwable exception){  for (int i = 0, s = registrants.size(); i < s ; i++) {  Registrant r = (Registrant) registrants.get(i);  r.internalNotifyRegistrant(result, exception);  }  }  /\*package\*/ 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);  h.sendMessage(msg);  }  } |

注册为观察者的方法为：

|  |
| --- |
| // Inherited documentation suffices.  @Override  public void registerForNewRingingConnection(  Handler h, int what, Object obj) {  checkCorrectThread(h);  mNewRingingConnectionRegistrants.addUnique(h, what, obj);  } |

通过log发现PstnIncomingCallNotifier.java调用registerForNewRingingConnection()

# 第二部分：PstnIncomingCallNotifier–>Call 接收Framework层到通知–>准备创建连接

## 7、PstnIncomingCallNotifier类对来电消息的处理

packages/services/Telephony/…/PstnIncomingCallNotifier.java（packages\services\telephony）

作用：监听来之相关电话对象的来电事件和通知Telecom在每次发生的时候，这一实例的存在为了每个电话的通话服务registerForNotifications方法调用registerForNewRingingConnection

|  |
| --- |
| private void registerForNotifications() {  Phone newPhone = mPhoneProxy.getActivePhone();  if (newPhone != mPhoneBase) {  unregisterForNotifications();  if (newPhone != null) {  Log.i(this, "Registering: %s", newPhone);  mPhoneBase = newPhone;  //调用registerForNewRingingConnection方法  mPhoneBase.registerForNewRingingConnection(  mHandler, EVENT\_NEW\_RINGING\_CONNECTION, null);  mPhoneBase.registerForCallWaiting(  mHandler, EVENT\_CDMA\_CALL\_WAITING, null);  mPhoneBase.registerForUnknownConnection(mHandler, EVENT\_UNKNOWN\_CONNECTION,  null);  }  }  } |

handle 处理EVENT\_NEW\_RINGING\_CONNECTION消息

|  |
| --- |
| private final [Handler](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=Handler&project=odm-foxconn-android-m-kanuti-tulip-somc) [mHandler](http://opengrok.sonymobile.net/JDM-BRANCHES/s?refs=mHandler&project=odm-foxconn-android-m-kanuti-tulip-somc) = new [Handler](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=Handler&project=odm-foxconn-android-m-kanuti-tulip-somc)() {  @[Override](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=Override&project=odm-foxconn-android-m-kanuti-tulip-somc)  public void [handleMessage](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=handleMessage&project=odm-foxconn-android-m-kanuti-tulip-somc)([Message](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=Message&project=odm-foxconn-android-m-kanuti-tulip-somc) [msg](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=msg&project=odm-foxconn-android-m-kanuti-tulip-somc)) {  switch([msg](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=msg&project=odm-foxconn-android-m-kanuti-tulip-somc).[what](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=what&project=odm-foxconn-android-m-kanuti-tulip-somc)) {  case [EVENT\_NEW\_RINGING\_CONNECTION](http://opengrok.sonymobile.net/JDM-BRANCHES/xref/odm-foxconn-android-m-kanuti-tulip-somc/packages/services/Telephony/src/com/android/services/telephony/PstnIncomingCallNotifier.java#EVENT_NEW_RINGING_CONNECTION):  [handleNewRingingConnection](http://opengrok.sonymobile.net/JDM-BRANCHES/xref/odm-foxconn-android-m-kanuti-tulip-somc/packages/services/Telephony/src/com/android/services/telephony/PstnIncomingCallNotifier.java#handleNewRingingConnection)(([AsyncResult](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=AsyncResult&project=odm-foxconn-android-m-kanuti-tulip-somc)) [msg](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=msg&project=odm-foxconn-android-m-kanuti-tulip-somc).[obj](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=obj&project=odm-foxconn-android-m-kanuti-tulip-somc));  break;  case [EVENT\_CDMA\_CALL\_WAITING](http://opengrok.sonymobile.net/JDM-BRANCHES/xref/odm-foxconn-android-m-kanuti-tulip-somc/packages/services/Telephony/src/com/android/services/telephony/PstnIncomingCallNotifier.java#EVENT_CDMA_CALL_WAITING):  [handleCdmaCallWaiting](http://opengrok.sonymobile.net/JDM-BRANCHES/xref/odm-foxconn-android-m-kanuti-tulip-somc/packages/services/Telephony/src/com/android/services/telephony/PstnIncomingCallNotifier.java#handleCdmaCallWaiting)(([AsyncResult](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=AsyncResult&project=odm-foxconn-android-m-kanuti-tulip-somc)) [msg](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=msg&project=odm-foxconn-android-m-kanuti-tulip-somc).[obj](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=obj&project=odm-foxconn-android-m-kanuti-tulip-somc));  break;  case [EVENT\_UNKNOWN\_CONNECTION](http://opengrok.sonymobile.net/JDM-BRANCHES/xref/odm-foxconn-android-m-kanuti-tulip-somc/packages/services/Telephony/src/com/android/services/telephony/PstnIncomingCallNotifier.java#EVENT_UNKNOWN_CONNECTION):  [handleNewUnknownConnection](http://opengrok.sonymobile.net/JDM-BRANCHES/xref/odm-foxconn-android-m-kanuti-tulip-somc/packages/services/Telephony/src/com/android/services/telephony/PstnIncomingCallNotifier.java#handleNewUnknownConnection)(([AsyncResult](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=AsyncResult&project=odm-foxconn-android-m-kanuti-tulip-somc)) [msg](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=msg&project=odm-foxconn-android-m-kanuti-tulip-somc).[obj](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=obj&project=odm-foxconn-android-m-kanuti-tulip-somc));  break;  default:  break;  }  }  }; |

handleNewRingingConnection方法处理新的来电连接。

|  |
| --- |
| private void handleNewRingingConnection(AsyncResult asyncResult) {  Log.d(this, "handleNewRingingConnection");  Connection connection = (Connection) asyncResult.result;//将消息转化为Connection对象  if (connection != null) {  Call call = connection.getCall();//通过connection对象获得Call  //在发送intent到Telecom之前最后一次验证ringing 状态  if ([call](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=call&project=odm-foxconn-android-m-kanuti-tulip-somc) != [null](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=null&project=odm-foxconn-android-m-kanuti-tulip-somc) && [call](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=call&project=odm-foxconn-android-m-kanuti-tulip-somc).[getState](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=getState&project=odm-foxconn-android-m-kanuti-tulip-somc)().[isRinging](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=isRinging&project=odm-foxconn-android-m-kanuti-tulip-somc)()) {  boolean [dropCall](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=dropCall&project=odm-foxconn-android-m-kanuti-tulip-somc) = false;  // Check if call should be dropped due to Voice Roaming policy  [DevicePolicyManager](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=DevicePolicyManager&project=odm-foxconn-android-m-kanuti-tulip-somc) [dpm](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=dpm&project=odm-foxconn-android-m-kanuti-tulip-somc) = ([DevicePolicyManager](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=DevicePolicyManager&project=odm-foxconn-android-m-kanuti-tulip-somc))[mPhoneProxy](http://opengrok.sonymobile.net/JDM-BRANCHES/xref/odm-foxconn-android-m-kanuti-tulip-somc/packages/services/Telephony/src/com/android/services/telephony/PstnIncomingCallNotifier.java#mPhoneProxy).[getContext](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=getContext&project=odm-foxconn-android-m-kanuti-tulip-somc)().  [getSystemService](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=getSystemService&project=odm-foxconn-android-m-kanuti-tulip-somc)([Context](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=Context&project=odm-foxconn-android-m-kanuti-tulip-somc).[DEVICE\_POLICY\_SERVICE](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=DEVICE_POLICY_SERVICE&project=odm-foxconn-android-m-kanuti-tulip-somc));  if ([dpm](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=dpm&project=odm-foxconn-android-m-kanuti-tulip-somc) != [null](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=null&project=odm-foxconn-android-m-kanuti-tulip-somc) && [dpm](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=dpm&project=odm-foxconn-android-m-kanuti-tulip-somc).[isVoiceRoamingDisabled](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=isVoiceRoamingDisabled&project=odm-foxconn-android-m-kanuti-tulip-somc)([null](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=null&project=odm-foxconn-android-m-kanuti-tulip-somc))) {  [TelephonyManager](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=TelephonyManager&project=odm-foxconn-android-m-kanuti-tulip-somc) [tm](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=tm&project=odm-foxconn-android-m-kanuti-tulip-somc) = ([TelephonyManager](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=TelephonyManager&project=odm-foxconn-android-m-kanuti-tulip-somc))[mPhoneProxy](http://opengrok.sonymobile.net/JDM-BRANCHES/xref/odm-foxconn-android-m-kanuti-tulip-somc/packages/services/Telephony/src/com/android/services/telephony/PstnIncomingCallNotifier.java#mPhoneProxy).[getContext](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=getContext&project=odm-foxconn-android-m-kanuti-tulip-somc)().  [getSystemService](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=getSystemService&project=odm-foxconn-android-m-kanuti-tulip-somc)([Context](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=Context&project=odm-foxconn-android-m-kanuti-tulip-somc).[TELEPHONY\_SERVICE](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=TELEPHONY_SERVICE&project=odm-foxconn-android-m-kanuti-tulip-somc));  if ([tm](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=tm&project=odm-foxconn-android-m-kanuti-tulip-somc) != [null](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=null&project=odm-foxconn-android-m-kanuti-tulip-somc)) {  [dropCall](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=dropCall&project=odm-foxconn-android-m-kanuti-tulip-somc) = [tm](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=tm&project=odm-foxconn-android-m-kanuti-tulip-somc).[isNetworkRoaming](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=isNetworkRoaming&project=odm-foxconn-android-m-kanuti-tulip-somc)([call](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=call&project=odm-foxconn-android-m-kanuti-tulip-somc).[getPhone](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=getPhone&project=odm-foxconn-android-m-kanuti-tulip-somc)().[getSubId](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=getSubId&project=odm-foxconn-android-m-kanuti-tulip-somc)());  }  }  if ([dropCall](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=dropCall&project=odm-foxconn-android-m-kanuti-tulip-somc)) {//false  [Log](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=Log&project=odm-foxconn-android-m-kanuti-tulip-somc).i(this, "Call dropped due to Voice Roaming policy");  try {  [call](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=call&project=odm-foxconn-android-m-kanuti-tulip-somc).[hangup](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=hangup&project=odm-foxconn-android-m-kanuti-tulip-somc)();  } catch ([CallStateException](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=CallStateException&project=odm-foxconn-android-m-kanuti-tulip-somc) e) {  [Log](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=Log&project=odm-foxconn-android-m-kanuti-tulip-somc).e(this, e, "Call hangup failed with exception");  }  } else {  [sendIncomingCallIntent](http://opengrok.sonymobile.net/JDM-BRANCHES/xref/odm-foxconn-android-m-kanuti-tulip-somc/packages/services/Telephony/src/com/android/services/telephony/PstnIncomingCallNotifier.java#sendIncomingCallIntent)([connection](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=connection&project=odm-foxconn-android-m-kanuti-tulip-somc));//将Intent发送到Telecom  }  }  }  } |

sendIncomingCallIntent方法实现

发送incoming call intent到telecom，发送的Connection 类型，里面包括isIncoming getState isRinging等

|  |
| --- |
| /\*\*  \* Sends the incoming call intent to telecom.  \*/  private void sendIncomingCallIntent(Connection connection) {  Bundle extras = null;  if (connection.getNumberPresentation() == TelecomManager.PRESENTATION\_ALLOWED &&  !TextUtils.isEmpty(connection.getAddress())) {  extras = new Bundle();  Uri uri = Uri.fromParts(PhoneAccount.SCHEME\_TEL, connection.getAddress(), null);  extras.putParcelable(TelephonyManager.EXTRA\_INCOMING\_NUMBER, uri);  }  TelecomManager.from(mPhoneProxy.getContext()).addNewIncomingCall(  TelecomAccountRegistry.makePstnPhoneAccountHandle(mPhoneProxy), extras);  } |

addNewIncomingCall()方法实现

addNewIncomingCall()定义在： framworks/base/telecomm/java/android/telecom/TelecomManager.java

作用：TelecomManager的功能则主要是对TelecomService提供的远程接口的封装，然后提供给应用使用。  
来电时触发 addNewIncomingCall方法

|  |
| --- |
| @SystemApi  public void addNewIncomingCall(PhoneAccountHandle phoneAccount, Bundle extras) {  try {  if (isServiceConnected()) {  getTelecomService().addNewIncomingCall(  phoneAccount, extras == null ? new Bundle() : extras);  }  } catch (RemoteException e) {  Log.e(TAG, "RemoteException adding a new incoming call: " + phoneAccount, e);  }  }  private ITelecomService getTelecomService() {  return ITelecomService.Stub.asInterface(ServiceManager.getService(Context.TELECOM\_SERVICE));  } |

getTelecomService()得到的是ITelecomService.aidl的实现类的对象，其实现类为TelecomServiceImpl.java

## 8、 addNewIncomingCall方法在TelecomServiceImpl.java的实现

packages/services/Telecomm/src/com/android/server/telecom/TelecomServiceImpl.java  
addNewIncomingCall方法中会新建intent 设定intent 的ACTION 、addFalgs等，

|  |
| --- |
| private final ITelecomService.Stub mBinderImpl = new ITelecomService.Stub() {  /\*\*  \* @see android.telecom.TelecomManager#addNewIncomingCall  \*/  @Override  public void addNewIncomingCall(PhoneAccountHandle phoneAccountHandle, Bundle extras) {  synchronized (mLock) {  Log.i(this, "Adding new incoming call with phoneAccountHandle %s",  phoneAccountHandle);// ComponentInfo{com.android.phone/com.android.services.telephony.  //TelephonyConnectionService}, 1, UserHandle{0}  if (phoneAccountHandle != null && phoneAccountHandle.getComponentName() != null) {  // TODO(sail): Add unit tests for adding incoming calls from a SIM call manager.  if (isCallerSimCallManager() && TelephonyUtil.isPstnComponentName(  phoneAccountHandle.getComponentName())) {  Log.v(this, "Allowing call manager to add incoming call with PSTN handle");  } else {  mAppOpsManager.checkPackage(  Binder.getCallingUid(),  phoneAccountHandle.getComponentName().getPackageName());  // Make sure it doesn't cross the UserHandle boundary  enforceUserHandleMatchesCaller(phoneAccountHandle);  enforcePhoneAccountIsRegisteredEnabled(phoneAccountHandle);  }  long token = Binder.clearCallingIdentity();  try {  Intent intent = new Intent(TelecomManager.ACTION\_INCOMING\_CALL);  intent.putExtra(TelecomManager.EXTRA\_PHONE\_ACCOUNT\_HANDLE,  phoneAccountHandle);  intent.putExtra(CallIntentProcessor.KEY\_IS\_INCOMING\_CALL, true);  if (extras != null) {  intent.putExtra(TelecomManager.EXTRA\_INCOMING\_CALL\_EXTRAS, extras);  }  CallIntentProcessor.processIncomingCallIntent(mCallsManager, intent);  } finally {  Binder.restoreCallingIdentity(token);  }  } else {  Log.w(this,  "Null phoneAccountHandle. Ignoring request to add new incoming call");  }  }  }  } |

## 9、 processIncomingCallIntent 方法在CallIntentProcessor.java 的实现

CallIntentProcessor.java (packages\services\telecomm\src\com\android\server\telecom).

通过调用processIncomingCallIntent方法将intent作为参数传过来，并在processIncomingCallIntent方法中去判断是否是设备的持有者

|  |
| --- |
| static void processIncomingCallIntent(CallsManager callsManager, Intent intent) {  PhoneAccountHandle phoneAccountHandle = intent.getParcelableExtra(  TelecomManager.EXTRA\_PHONE\_ACCOUNT\_HANDLE);  if (phoneAccountHandle == null) {  Log.w(CallIntentProcessor.class,  "Rejecting incoming call due to null phone account");  return;  }  if (phoneAccountHandle.getComponentName() == null) {  Log.w(CallIntentProcessor.class,  "Rejecting incoming call due to null component name");  return;  }  Bundle clientExtras = null;  if (intent.hasExtra(TelecomManager.EXTRA\_INCOMING\_CALL\_EXTRAS)) {  clientExtras = intent.getBundleExtra(TelecomManager.EXTRA\_INCOMING\_CALL\_EXTRAS);  }  if (clientExtras == null) {  clientExtras = new Bundle();  }  Log.d(CallIntentProcessor.class,  "Processing incoming call from connection service [%s]",  phoneAccountHandle.getComponentName());  callsManager.processIncomingCallIntent(phoneAccountHandle, clientExtras);  } |

## 10、processIncomingCallIntent方法在CallsManager类的实现

packages/services/Telecomm/src/com/android/server/telecom/CallsManager.java

作用:  
CallManager类提供了一个抽象层，以供phoneApp访问和控制call等操作。它实现了Phone接口。CallManager提供呼叫和连接控制以及Channel能力。CallManager提供三种类型的API：  
1，呼叫控制和操作，如dial()和hangup()；  
2，Channel的能力，如canconference()；  
3，注册通知。接着将Phone注册进mCM,Phone状态改变之后InCallUI就能够收到变化消息了。

|  |
| --- |
| void processIncomingCallIntent(PhoneAccountHandle phoneAccountHandle, Bundle extras) {  Uri handle = extras.getParcelable(TelecomManager.EXTRA\_INCOMING\_CALL\_ADDRESS);  if (handle == null) {  // Required for backwards compatibility  handle = extras.getParcelable(TelephonyManager.EXTRA\_INCOMING\_NUMBER);  }  Call call = new Call(  mContext,  this,  mLock,  mConnectionServiceRepository,  mContactsAsyncHelper,  mCallerInfoAsyncQueryFactory,  handle,  null /\* gatewayInfo \*/,  null /\* connectionManagerPhoneAccount \*/,  phoneAccountHandle,  true /\* isIncoming \*/,  false /\* isConference \*/);  call.setIntentExtras(extras);  call.addListener(this);  call.startCreateConnection(mPhoneAccountRegistrar);  // new一个Call 对象 把前面的参数传进来，然后调用call中建立连接的方法startCreateConnection  } |

## 11、startCreateConnection 在Call.java的实现

packages/services/Telecomm/src/com/android/server/telecom/Call.java

作用：封装一个给定的电话在其整个生命周期的各个方面，从电话意图被telecom接收开始  
开始建立连接队列，一旦完成创建，就应当有一个活动active的连接了存在service里。

|  |
| --- |
| void startCreateConnection(PhoneAccountRegistrar phoneAccountRegistrar) {  Preconditions.checkState(mCreateConnectionProcessor == null);  mCreateConnectionProcessor = new CreateConnectionProcessor(this, mRepository, this,  phoneAccountRegistrar, mContext);  mCreateConnectionProcessor.process();  } |

process方法的实现

packages/services/Telecomm/src/com/android/server/telecom/CreateConnectionProcessor.java

|  |
| --- |
| CreateConnectionProcessor(  Call call, ConnectionServiceRepository repository, CreateConnectionResponse response,  PhoneAccountRegistrar phoneAccountRegistrar, Context context) {  Log.v(this, "CreateConnectionProcessor created for Call = %s", call);  mCall = call;  mRepository = repository;  mResponse = response;  mPhoneAccountRegistrar = phoneAccountRegistrar;  mContext = context;  }  void process() {  Log.v(this, "process");  clearTimeout();  mAttemptRecords = new ArrayList<>();  if (mCall.getTargetPhoneAccount() != null) {  mAttemptRecords.add(new CallAttemptRecord(  mCall.getTargetPhoneAccount(), mCall.getTargetPhoneAccount()));  }  adjustAttemptsForConnectionManager();  adjustAttemptsForEmergency();  mAttemptRecordIterator = mAttemptRecords.iterator();  attemptNextPhoneAccount();  } |

attemptNextPhoneAccount方法的实现，service试图建立连接

|  |
| --- |
| private void attemptNextPhoneAccount() {  ...  if (mResponse != null && attempt != null) {  Log.i(this, "Trying attempt %s", attempt);  ConnectionServiceWrapper service =  mRepository.getService(  attempt.connectionManagerPhoneAccount.getComponentName());  if (service == null) {  Log.i(this, "Found no connection service for attempt %s", attempt);  attemptNextPhoneAccount();  } else {  mCall.setConnectionManagerPhoneAccount(attempt.connectionManagerPhoneAccount);  mCall.setTargetPhoneAccount(attempt.targetPhoneAccount);  mCall.setConnectionService(service);  Log.i(this, "Attempting to call from %s", service.getComponentName());  service.createConnection(mCall, new Response(service));  }  } |

createConnection方法在ConnectionServiceWrapper.java的实现

packages/services/Telecomm/src/com/android/server/telecom/ConnectionServiceWrapper.java

为拨出的电话建立连接，或者attach一个已经存在的来电。

|  |
| --- |
| void createConnection(final Call call, final CreateConnectionResponse response) {  // callback接口的实现  mBinder.bind(callback);//启动service  } |

## 12、bind()方法的实现（绑定服务）

ServiceBinder.java (packages\services\telecomm\src\com\android\server\telecom)

作用：抽象类用来进行绑定和解除绑定到指定的服务接口的工作。子类提供服务的意图和组件名称和这个类调用受保护方法在类绑定，未绑定或者失败的时候  
执行绑定到服务的操作（如果还没有绑定）然后执行指定的回调方法

|  |
| --- |
| void bind(BindCallback callback, Call call) {  Log.d(ServiceBinder.this, "bind()");  // Reset any abort request if we're asked to bind again.  clearAbort();  if (!mCallbacks.isEmpty()) {  // Binding already in progress, append to the list of callbacks and bail out.  mCallbacks.add(callback);  return;  }  mCallbacks.add(callback);  if (mServiceConnection == null) {//首次来电时会绑定相应的服务  Intent serviceIntent = new Intent(mServiceAction).setComponent(mComponentName);  ServiceConnection connection = new ServiceBinderConnection(call);  Log.event(call, Log.Events.BIND\_CS, mComponentName);  final int bindingFlags = Context.BIND\_AUTO\_CREATE | Context.BIND\_FOREGROUND\_SERVICE;  final boolean isBound;  if (mUserHandle != null) {  isBound = mContext.bindServiceAsUser(serviceIntent, connection, bindingFlags,  mUserHandle);  } else {  isBound = mContext.bindService(serviceIntent, connection, bindingFlags);  }  if (!isBound) {  handleFailedConnection();  return;  }  } else {  Log.d(ServiceBinder.this, "Service is already bound.");  Preconditions.checkNotNull(mBinder);  handleSuccessfulConnection();  }  } |

上面的执行完之后，顺序执行到onServiceConnected

|  |
| --- |
| private final class ServiceBinderConnection implements ServiceConnection {  /\*\*  \* The initial call for which the service was bound.  \*/  private Call mCall;  ServiceBinderConnection(Call call) {  mCall = call;  }  @Override  public void onServiceConnected(ComponentName componentName, IBinder binder) {  synchronized (mLock) {  Log.i(this, "Service bound %s", componentName);  Log.event(mCall, Log.Events.CS\_BOUND, componentName);  mCall = null;  // Unbind request was queued so unbind immediately.  if (mIsBindingAborted) {  clearAbort();  logServiceDisconnected("onServiceConnected");  mContext.unbindService(this);  handleFailedConnection();  return;  }  mServiceConnection = this;  setBinder(binder);  handleSuccessfulConnection();  }  }  } |

## 13、callback接口回调中的onSuccess()方法的实现

通过call对象将相应的通话信息（来电电话号码，归属地等）传输给可与Android设备进行连接的服务进程。

|  |
| --- |
| private void handleSuccessfulConnection() {  for (BindCallback callback : mCallbacks) {  callback.onSuccess();  }  mCallbacks.clear();  }  void createConnection(final Call call, final CreateConnectionResponse response) {  Log.d(this, "createConnection(%s) via %s.", call, getComponentName());  BindCallback callback = new BindCallback() {  @Override  public void onSuccess() {  String callId = mCallIdMapper.getCallId(call);//1  mPendingResponses.put(callId, response);  GatewayInfo gatewayInfo = call.getGatewayInfo();  Bundle extras = call.getIntentExtras();  if (gatewayInfo != null && gatewayInfo.getGatewayProviderPackageName() != null &&  gatewayInfo.getOriginalAddress() != null) {  extras = (Bundle) extras.clone();  extras.putString(  TelecomManager.GATEWAY\_PROVIDER\_PACKAGE,  gatewayInfo.getGatewayProviderPackageName());  extras.putParcelable(  TelecomManager.GATEWAY\_ORIGINAL\_ADDRESS,  gatewayInfo.getOriginalAddress());  }  Log.event(call, Log.Events.START\_CONNECTION, Log.piiHandle(call.getHandle()));  try {  mServiceInterface.createConnection(  call.getConnectionManagerPhoneAccount(),  callId,  new ConnectionRequest(  call.getTargetPhoneAccount(),  call.getHandle(),  extras,  call.getVideoState()),  call.isIncoming(),  call.isUnknown());  } catch (RemoteException e) {  Log.e(this, e, "Failure to createConnection -- %s", getComponentName());  mPendingResponses.remove(callId).handleCreateConnectionFailure(  new DisconnectCause(DisconnectCause.ERROR, e.toString()));  }  }  @Override  public void onFailure() {  Log.e(this, new Exception(), "Failure to call %s", getComponentName());  response.handleCreateConnectionFailure(new DisconnectCause(DisconnectCause.ERROR));  }  };  mBinder.bind(callback, call);  } |

# 第三部分：ConnectionService–> ConnectionServicesAdapter–>CallsManager 处理这个创建的连接–>成功来电 CallsManager–>Phone 成功来电–>准备启动界面

## 14、createConnection方法在ConnectionService类的实现

ConnectionService.java (frameworks\base\telecomm\java\android\telecom)

createConnection这个方法可以被telecom用来创建呼出电话或者一个已存在的来电。任何一种情况，telecom都会循环经过一系列的服务和 调用createConnection util a connection service取消或者成功完成创建

作用：ConnectionService是一个提供电话连接到Android设备上运行的服务进程。

ConnectionServiceAdapter>CallsManager 处理这个创建的连接>成功来电

|  |
| --- |
| private void createConnection(  final PhoneAccountHandle callManagerAccount,  final String callId,  final ConnectionRequest request,  boolean isIncoming,  boolean isUnknown) {  Log.d(this, "createConnection, callManagerAccount: %s, callId: %s, request: %s, " +  "isIncoming: %b, isUnknown: %b", callManagerAccount, callId, request, isIncoming,  isUnknown);  //ComponentInfo{com.android.phone/com.android.services.telephony.TelephonyConnectionService}, 1, UserHandle{0},  //callId: ConnectionService@1, request: ConnectionRequest tel:17718383527 Bundle[mParcelledData.dataSize=216], isIncoming: true, isUnknown: false  Connection connection = isUnknown ? onCreateUnknownConnection(callManagerAccount, request)  : isIncoming ? onCreateIncomingConnection(callManagerAccount, request)  : onCreateOutgoingConnection(callManagerAccount, request);  Log.d(this, "createConnection, connection: %s", connection);  if (connection == null) {  connection = Connection.createFailedConnection(  new DisconnectCause(DisconnectCause.ERROR));  }  if (connection.getState() != Connection.STATE\_DISCONNECTED) {  addConnection(callId, connection);  }  Uri address = connection.getAddress();  String number = address == null ? "null" : address.getSchemeSpecificPart();  Log.v(this, "createConnection, number: %s, state: %s, capabilities: %s",  Connection.toLogSafePhoneNumber(number),  Connection.stateToString(connection.getState()),  Connection.capabilitiesToString(connection.getConnectionCapabilities()));  //createConnection, number: 17718383527, state: RINGING, capabilities:  //[Capabilities: CAPABILITY\_SUPPORT\_HOLD CAPABILITY\_MUTE CAPABILITY\_SPEED\_UP\_MT\_AUDIO]  Log.d(this, "createConnection, calling handleCreateConnectionSuccessful %s", callId);//ConnectionService@1  Log.d(this, "[WP1] connection.codec = %s",connection.getCodec()); //SW12-JC-HUTCH\_HD\_VOICE\_INDICATOR-01+  mAdapter.handleCreateConnectionComplete(  callId,  request,  new ParcelableConnection(  request.getAccountHandle(),  connection.getState(),  connection.getConnectionCapabilities(),  connection.getAddress(),  connection.getAddressPresentation(),  connection.getCallerDisplayName(),  connection.getCallerDisplayNamePresentation(),  connection.getCodec(), //SW12-JC-HUTCH\_HD\_VOICE\_INDICATOR-01+  connection.getVideoProvider() == null ?  null : connection.getVideoProvider().getInterface(),  connection.getVideoState(),  connection.isRingbackRequested(),  connection.getAudioModeIsVoip(),  connection.getConnectTimeMillis(),  connection.getStatusHints(),  connection.getDisconnectCause(),  createIdList(connection.getConferenceables()),  connection.getExtras()));  if (isUnknown) {  triggerConferenceRecalculate();  }  } |

## 15、handleCreateConnectionComplete 在ConnectionServiceAdapter类的实现

ConnectionServiceAdapter.java (frameworks\base\telecomm\java\android\telecom)

作用：提供iconnectionservice实现与系统的手机应用程序的交互方法。

前面建立连接成功了，后面处理成功的连接，后面执行mAdapter.handleCreateConnectionComplete

|  |
| --- |
| void handleCreateConnectionComplete(  String id,  ConnectionRequest request,  ParcelableConnection connection) {  for (IConnectionServiceAdapter adapter : mAdapters) {  try {  adapter.handleCreateConnectionComplete(id, request, connection);  } catch (RemoteException e) {  }  }  } |

// IConnectionServiceAdapter是一个aidl其实现的类有ConnectionService.java 、ConnectionServiceAdapter.java、 ConnectionServiceWrapper.java

//此时的handleCreateConnectionComplete方法的实现是在ConnectionServiceWrapper类中。他是Telecomm 层的连接管理者类

packages/services/Telecomm/src/com/android/server/telecom/ConnectionServiceWrapper.java

|  |
| --- |
| private void handleCreateConnectionComplete(  String callId,  ConnectionRequest request,  ParcelableConnection connection) {  // TODO: Note we are not using parameter "request", which is a side effect of our tacit  // assumption that we have at most one outgoing connection attempt per ConnectionService.  // This may not continue to be the case.  if (connection.getState() == Connection.STATE\_DISCONNECTED) {  //当连接connection断开时,从callList集合中移除call  // A connection that begins in the DISCONNECTED state is an indication of  // failure to connect; we handle all failures uniformly  removeCall(callId, connection.getDisconnectCause());  } else {  // Successful connection  if (mPendingResponses.containsKey(callId)) {  mPendingResponses.remove(callId)  .handleCreateConnectionSuccess(mCallIdMapper, connection);  }  }  } |

## 16、 handleCreateConnectionComplete在ConnectionServiceAdapterServant类中连接成功的消息处理

ConnectionServiceAdapterServant.java (frameworks\base\telecomm\java\android\telecom)

|  |
| --- |
| private final IConnectionServiceAdapter mStub = new IConnectionServiceAdapter.Stub() {  @Override  public void handleCreateConnectionComplete(  String id,  ConnectionRequest request,  ParcelableConnection connection) {  SomeArgs args = SomeArgs.obtain();  args.arg1 = id;  args.arg2 = request;  args.arg3 = connection;  mHandler.obtainMessage(MSG\_HANDLE\_CREATE\_CONNECTION\_COMPLETE, args).sendToTarget(); |

handleMessage处理消息 MSG\_HANDLE\_CREATE\_CONNECTION\_COMPLETE

|  |
| --- |
| // Internal method defined to centralize handling of RemoteException  private void internalHandleMessage(Message msg) throws RemoteException {  switch (msg.what) {  case MSG\_HANDLE\_CREATE\_CONNECTION\_COMPLETE: {  SomeArgs args = (SomeArgs) msg.obj;  try {  mDelegate.handleCreateConnectionComplete(  (String) args.arg1,  (ConnectionRequest) args.arg2,  (ParcelableConnection) args.arg3);  } finally {  args.recycle();  }  break;  } |

ConnectionServiceWrapper.java(packages\services\telecomm\src\com\android\server\telecom)

如果成功连接就会执行handleCreateConnectionComplete 方法

|  |
| --- |
| private void handleCreateConnectionComplete(  String callId,  ConnectionRequest request,  ParcelableConnection connection) {  // TODO: Note we are not using parameter "request", which is a side effect of our tacit  // assumption that we have at most one outgoing connection attempt per ConnectionService.  // This may not continue to be the case.  if (connection.getState() == Connection.STATE\_DISCONNECTED) {  // A connection that begins in the DISCONNECTED state is an indication of  // failure to connect; we handle all failures uniformly  removeCall(callId, connection.getDisconnectCause());  } else {  // Successful connection  if (mPendingResponses.containsKey(callId)) {  mPendingResponses.remove(callId)  .handleCreateConnectionSuccess(mCallIdMapper, connection);  }  }  } |

在Call.java类中重写 handleCreateConnectionSuccess方法

## 17、handleCreateConnectionSuccess方法在Call中的实现

packages/services/Telecomm/src/com/android/server/telecom/Call.java handleCreateConnetionSucess()

|  |
| --- |
| @Override  public void handleCreateConnectionSuccess(  CallIdMapper idMapper,  ParcelableConnection connection) {  } else if (mIsIncoming) {//true  // We do not handle incoming calls immediately when they are verified by the connection  // service. We allow the caller-info-query code to execute first so that we can read the  // direct-to-voicemail property before deciding if we want to show the incoming call to  // the user or if we want to reject the call.  mDirectToVoicemailQueryPending = true;  // Timeout the direct-to-voicemail lookup execution so that we dont wait too long before  // showing the user the incoming call screen.  mHandler.postDelayed(mDirectToVoicemailRunnable, Timeouts.getDirectToVoicemailMillis(  mContext.getContentResolver()));  //  } else {  for (Listener l : mListeners) {  l.onSuccessfulOutgoingCall(this,  getStateFromConnectionState(connection.getState()));  }  }  } |

Runnable mDirectToVoicemailRunnable的实现

|  |
| --- |
| private final Runnable mDirectToVoicemailRunnable = new Runnable() {  @Override  public void run() {  processDirectToVoicemail();  } |

## 18、processDirectToVoicemail方法的实现

通过Listener的实现类对象调用onSuccessfulIncomingCall方法

Listener是一个接口，它的主要实现类有：Call.java 、CallsManager.java

|  |
| --- |
| final class Call implements CreateConnectionResponse {  /\*\*  \* Listener for events on the call.  \*/  interface Listener {  void onSuccessfulIncomingCall(Call call);  ...  private void processDirectToVoicemail() {  if (mDirectToVoicemailQueryPending) {  if (mCallerInfo != null && mCallerInfo.shouldSendToVoicemail) {  Log.i(this, "Directing call to voicemail: %s.", this);  // TODO: Once we move State handling from CallsManager to Call, we  // will not need to set STATE\_RINGING state prior to calling reject.  setState(CallState.RINGING);  reject(false, null);  } else {  // TODO: Make this class (not CallsManager) responsible for changing  // the call state to STATE\_RINGING.  // TODO: Replace this with state transition to STATE\_RINGING.  for (Listener l : mListeners) {  l.onSuccessfulIncomingCall(this);  //通过Listener对象调用onSuccessfulIncomingCall方法  //Listener是一个接口，它的主要实现类有：Call.java 、CallsManager.java  }  }  mDirectToVoicemailQueryPending = false;  }  } |

CallsManager>Phone 成功来电>准备启动界面

## 19、 onSuccessfulIncomingCall方法在CallsManager类的实现

package/services/Telecomm/src/com/android/server/telecom/CallsManager.java

|  |
| --- |
| public final class CallsManager extends Call.ListenerBase {  ...  @Override  public void onSuccessfulIncomingCall(Call incomingCall) {  Log.d(this, "onSuccessfulIncomingCall");  setCallState(incomingCall, CallState.RINGING);//将call信息进行数据库存储  if (hasMaximumRingingCalls(incomingCall.getTargetPhoneAccount().getId())) {  incomingCall.reject(false, null);  // since the call was not added to the list of calls, we have to call the missed  // call notifier and the call logger manually.  mMissedCallNotifier.showMissedCallNotification(incomingCall);  mCallLogManager.logCall(incomingCall, Calls.MISSED\_TYPE);  } else {  incomingCall.mIsActiveSub = true;  addCall(incomingCall);  setActiveSubscription(incomingCall.getTargetPhoneAccount().getId());  }  } |

addCall()将来电添加到Listener监听中

|  |
| --- |
| [1641](http://opengrok.sonymobile.net/JDM-BRANCHES/xref/odm-foxconn-android-m-kanuti-tulip-somc/packages/services/Telecomm/src/com/android/server/telecom/CallsManager.java?a=true&h=" \l "1641) private void [addCall](http://opengrok.sonymobile.net/JDM-BRANCHES/s?refs=addCall&project=odm-foxconn-android-m-kanuti-tulip-somc)([Call](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=Call&project=odm-foxconn-android-m-kanuti-tulip-somc) [call](http://opengrok.sonymobile.net/JDM-BRANCHES/s?refs=call&project=odm-foxconn-android-m-kanuti-tulip-somc)) {  [1642](http://opengrok.sonymobile.net/JDM-BRANCHES/xref/odm-foxconn-android-m-kanuti-tulip-somc/packages/services/Telecomm/src/com/android/server/telecom/CallsManager.java?a=true&h=" \l "1642) [Trace](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=Trace&project=odm-foxconn-android-m-kanuti-tulip-somc).[beginSection](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=beginSection&project=odm-foxconn-android-m-kanuti-tulip-somc)("addCall");  [1643](http://opengrok.sonymobile.net/JDM-BRANCHES/xref/odm-foxconn-android-m-kanuti-tulip-somc/packages/services/Telecomm/src/com/android/server/telecom/CallsManager.java?a=true&h=" \l "1643) [Log](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=Log&project=odm-foxconn-android-m-kanuti-tulip-somc).v(this, "addCall(%s)", [call](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=call&project=odm-foxconn-android-m-kanuti-tulip-somc));  [1644](http://opengrok.sonymobile.net/JDM-BRANCHES/xref/odm-foxconn-android-m-kanuti-tulip-somc/packages/services/Telecomm/src/com/android/server/telecom/CallsManager.java?a=true&h=" \l "1644) [call](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=call&project=odm-foxconn-android-m-kanuti-tulip-somc).[addListener](http://opengrok.sonymobile.net/JDM-BRANCHES/xref/odm-foxconn-android-m-kanuti-tulip-somc/packages/services/Telecomm/src/com/android/server/telecom/CallsManager.java?a=true&h=#addListener)(this);//给call添加Listener监听  [1645](http://opengrok.sonymobile.net/JDM-BRANCHES/xref/odm-foxconn-android-m-kanuti-tulip-somc/packages/services/Telecomm/src/com/android/server/telecom/CallsManager.java?a=true&h=" \l "1645) [mCalls](http://opengrok.sonymobile.net/JDM-BRANCHES/xref/odm-foxconn-android-m-kanuti-tulip-somc/packages/services/Telecomm/src/com/android/server/telecom/CallsManager.java?a=true&h=#mCalls).[add](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=add&project=odm-foxconn-android-m-kanuti-tulip-somc)([call](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=call&project=odm-foxconn-android-m-kanuti-tulip-somc));//将call添加到call List  [1646](http://opengrok.sonymobile.net/JDM-BRANCHES/xref/odm-foxconn-android-m-kanuti-tulip-somc/packages/services/Telecomm/src/com/android/server/telecom/CallsManager.java?a=true&h=" \l "1646)  [1647](http://opengrok.sonymobile.net/JDM-BRANCHES/xref/odm-foxconn-android-m-kanuti-tulip-somc/packages/services/Telecomm/src/com/android/server/telecom/CallsManager.java?a=true&h=" \l "1647) // TODO: Update mForegroundCall prior to invoking  [1648](http://opengrok.sonymobile.net/JDM-BRANCHES/xref/odm-foxconn-android-m-kanuti-tulip-somc/packages/services/Telecomm/src/com/android/server/telecom/CallsManager.java?a=true&h=" \l "1648) // onCallAdded for calls which immediately take the foreground (like the first call).  [1649](http://opengrok.sonymobile.net/JDM-BRANCHES/xref/odm-foxconn-android-m-kanuti-tulip-somc/packages/services/Telecomm/src/com/android/server/telecom/CallsManager.java?a=true&h=" \l "1649) for ([CallsManagerListener](http://opengrok.sonymobile.net/JDM-BRANCHES/xref/odm-foxconn-android-m-kanuti-tulip-somc/packages/services/Telecomm/src/com/android/server/telecom/CallsManager.java?a=true&h=#CallsManagerListener) [listener](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=listener&project=odm-foxconn-android-m-kanuti-tulip-somc) : [mListeners](http://opengrok.sonymobile.net/JDM-BRANCHES/xref/odm-foxconn-android-m-kanuti-tulip-somc/packages/services/Telecomm/src/com/android/server/telecom/CallsManager.java?a=true&h=#mListeners)) {  [1650](http://opengrok.sonymobile.net/JDM-BRANCHES/xref/odm-foxconn-android-m-kanuti-tulip-somc/packages/services/Telecomm/src/com/android/server/telecom/CallsManager.java?a=true&h=" \l "1650) if ([Log](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=Log&project=odm-foxconn-android-m-kanuti-tulip-somc).[SYSTRACE\_DEBUG](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=SYSTRACE_DEBUG&project=odm-foxconn-android-m-kanuti-tulip-somc)) {  [1651](http://opengrok.sonymobile.net/JDM-BRANCHES/xref/odm-foxconn-android-m-kanuti-tulip-somc/packages/services/Telecomm/src/com/android/server/telecom/CallsManager.java?a=true&h=" \l "1651) [Trace](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=Trace&project=odm-foxconn-android-m-kanuti-tulip-somc).[beginSection](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=beginSection&project=odm-foxconn-android-m-kanuti-tulip-somc)([listener](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=listener&project=odm-foxconn-android-m-kanuti-tulip-somc).[getClass](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=getClass&project=odm-foxconn-android-m-kanuti-tulip-somc)().[toString](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=toString&project=odm-foxconn-android-m-kanuti-tulip-somc)() + " addCall");  [1652](http://opengrok.sonymobile.net/JDM-BRANCHES/xref/odm-foxconn-android-m-kanuti-tulip-somc/packages/services/Telecomm/src/com/android/server/telecom/CallsManager.java?a=true&h=" \l "1652) }  [1653](http://opengrok.sonymobile.net/JDM-BRANCHES/xref/odm-foxconn-android-m-kanuti-tulip-somc/packages/services/Telecomm/src/com/android/server/telecom/CallsManager.java?a=true&h=" \l "1653) [listener](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=listener&project=odm-foxconn-android-m-kanuti-tulip-somc).[onCallAdded](http://opengrok.sonymobile.net/JDM-BRANCHES/xref/odm-foxconn-android-m-kanuti-tulip-somc/packages/services/Telecomm/src/com/android/server/telecom/CallsManager.java?a=true&h=#onCallAdded)([call](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=call&project=odm-foxconn-android-m-kanuti-tulip-somc));  [1654](http://opengrok.sonymobile.net/JDM-BRANCHES/xref/odm-foxconn-android-m-kanuti-tulip-somc/packages/services/Telecomm/src/com/android/server/telecom/CallsManager.java?a=true&h=" \l "1654) if ([Log](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=Log&project=odm-foxconn-android-m-kanuti-tulip-somc).[SYSTRACE\_DEBUG](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=SYSTRACE_DEBUG&project=odm-foxconn-android-m-kanuti-tulip-somc)) {  [1655](http://opengrok.sonymobile.net/JDM-BRANCHES/xref/odm-foxconn-android-m-kanuti-tulip-somc/packages/services/Telecomm/src/com/android/server/telecom/CallsManager.java?a=true&h=" \l "1655) [Trace](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=Trace&project=odm-foxconn-android-m-kanuti-tulip-somc).[endSection](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=endSection&project=odm-foxconn-android-m-kanuti-tulip-somc)();  [1656](http://opengrok.sonymobile.net/JDM-BRANCHES/xref/odm-foxconn-android-m-kanuti-tulip-somc/packages/services/Telecomm/src/com/android/server/telecom/CallsManager.java?a=true&h=" \l "1656) }  [1657](http://opengrok.sonymobile.net/JDM-BRANCHES/xref/odm-foxconn-android-m-kanuti-tulip-somc/packages/services/Telecomm/src/com/android/server/telecom/CallsManager.java?a=true&h=" \l "1657) }  [1658](http://opengrok.sonymobile.net/JDM-BRANCHES/xref/odm-foxconn-android-m-kanuti-tulip-somc/packages/services/Telecomm/src/com/android/server/telecom/CallsManager.java?a=true&h=" \l "1658) [updateCallsManagerState](http://opengrok.sonymobile.net/JDM-BRANCHES/xref/odm-foxconn-android-m-kanuti-tulip-somc/packages/services/Telecomm/src/com/android/server/telecom/CallsManager.java?a=true&h=#updateCallsManagerState)();//更新CallManager. State  [1659](http://opengrok.sonymobile.net/JDM-BRANCHES/xref/odm-foxconn-android-m-kanuti-tulip-somc/packages/services/Telecomm/src/com/android/server/telecom/CallsManager.java?a=true&h=" \l "1659) [Trace](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=Trace&project=odm-foxconn-android-m-kanuti-tulip-somc).[endSection](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=endSection&project=odm-foxconn-android-m-kanuti-tulip-somc)();  [1660](http://opengrok.sonymobile.net/JDM-BRANCHES/xref/odm-foxconn-android-m-kanuti-tulip-somc/packages/services/Telecomm/src/com/android/server/telecom/CallsManager.java?a=true&h=" \l "1660) } |

## 20、onCallAdded在InCallController类的实现

package/services/Telecomm/src/com/android/server/telecom/InCallController.java

作用：结合并提供服务，通过它可以将更新发送到呼叫程序。这类被创建和拥有的callsmanager保持绑定到（被调用的应用程序中实现）。

重写onCallAdded –>

|  |
| --- |
| @Override  public void onCallAdded(Call call) {  if (!isBoundToServices()) {  bindToServices(call);  } else {  adjustServiceBindingsForEmergency();  Log.i(this, "onCallAdded: %s", call);  // Track the call if we don't already know about it.  addCall(call);  for (Map.Entry entry : mInCallServices.entrySet()) {  ComponentName componentName = entry.getKey();  IInCallService inCallService = entry.getValue();  ParcelableCall parcelableCall = toParcelableCall(call,  true /\* includeVideoProvider \*/);  try {  inCallService.addCall(parcelableCall);  } catch (RemoteException ignored) {  }  }  }  } |

bindToServices – bindToInCallService开启InCallService服务

|  |
| --- |
| private void bindToServices(Call call) {  PackageManager packageManager = mContext.getPackageManager();  mInCallUIComponentName = inCallUIService;  // Bind to the control InCallServices  for (ComponentName componentName : inCallControlServices) {  bindToInCallService(componentName, call, "control");  }  }  private boolean bindToInCallService(ComponentName componentName, Call call, String tag) {  if (mInCallServices.containsKey(componentName)) {  Log.i(this, "An InCallService already exists: %s", componentName);  return true;  }  if (mServiceConnections.containsKey(componentName)) {  Log.w(this, "The service is already bound for this component %s", componentName);  return true;  }  Intent intent = new Intent(InCallService.SERVICE\_INTERFACE);  intent.setComponent(componentName);  if (call != null && !call.isIncoming()){  intent.putExtra(TelecomManager.EXTRA\_OUTGOING\_CALL\_EXTRAS,  call.getIntentExtras());  intent.putExtra(TelecomManager.EXTRA\_PHONE\_ACCOUNT\_HANDLE,  call.getTargetPhoneAccount());  }  Log.i(this, "Attempting to bind to [%s] InCall %s, with %s", tag, componentName, intent);  //Attempting to bind to [system]  //InCall ComponentInfo{com.android.incallui/com.android.incallui.InCallServiceImpl},  //with Intent { act=android.telecom.InCallService cmp=com.android.incallui/.InCallServiceImpl }  InCallServiceConnection inCallServiceConnection = new InCallServiceConnection();  if (mContext.bindServiceAsUser(intent, inCallServiceConnection,  Context.BIND\_AUTO\_CREATE | Context.BIND\_FOREGROUND\_SERVICE,  UserHandle.CURRENT)) {  mServiceConnections.put(componentName, inCallServiceConnection);  return true;  }  return false;  } |

InCallServiceConnection中执行onServiceConnected中的onConnected方法

|  |
| --- |
| InCallServiceConnection inCallServiceConnection = new InCallServiceConnection();  private class InCallServiceConnection implements ServiceConnection {  /\*\* {@inheritDoc} \*/  @Override public void onServiceConnected(ComponentName name, IBinder service) {  Log.d(this, "onServiceConnected: %s", name);  onConnected(name, service);  } |

执行完onServiceConnected后，继续执行onConnected方法

|  |
| --- |
| private void onConnected(ComponentName componentName, IBinder service) {  Trace.beginSection("onConnected: " + componentName);  Log.i(this, "onConnected to %s", componentName);  //ComponentInfo{com.android.incallui/com.android.incallui.InCallServiceImpl}  IInCallService inCallService = IInCallService.Stub.asInterface(service);  mInCallServices.put(componentName, inCallService);  try {  inCallService.setInCallAdapter(  new InCallAdapter(  mCallsManager,  mCallIdMapper,  mLock));  inCallService.setSomcInCallAdapterExtend(new SomcInCallAdapterExtend());//[TUL-23] LichiCLLi  } catch (RemoteException e) {  Log.e(this, e, "Failed to set the in-call adapter.");  Trace.endSection();  onInCallServiceFailure(componentName, "setInCallAdapter");  return;  }  // Upon successful connection, send the state of the world to the service.  Collection<Call> calls = mCallsManager.getCalls();  if (!calls.isEmpty()) {  Log.i(this, "Adding %s calls to InCallService after onConnected: %s", calls.size(),  componentName);  for (Call call : calls) {  try {  // Track the call if we don't already know about it.  addCall(call);//设置Call List的事件监听  inCallService.addCall(toParcelableCall(call, true /\* includeVideoProvider \*/));  } catch (RemoteException ignored) {  }  }  onCallAudioStateChanged(  null,  mCallsManager.getAudioState());  onCanAddCallChanged(mCallsManager.canAddCall());  } else {  unbindFromServices();  }  Trace.endSection();  } |

## 21、 addCall方法在InCallService方法中的实现

framworks/base/telecomm/java/android/telecom/InCallService.java

作用：这个服务可以被任何希望提供管理电话的用户界面的应用实现，  
当那个服务存在一个电话telecom就去绑定这个服务并用它去通知任何活动状态和最近断开的呼叫的被调用的应用

|  |
| --- |
| @Override addCall()  /\*\* Manages the binder calls so that the implementor does not need to deal with it. \*/  private final class InCallServiceBinder extends IInCallService.Stub {  @Override  public void setInCallAdapter(IInCallAdapter inCallAdapter) {  mHandler.obtainMessage(MSG\_SET\_IN\_CALL\_ADAPTER, inCallAdapter).sendToTarget();  }  @Override  public void addCall(ParcelableCall call) {  mHandler.obtainMessage(MSG\_ADD\_CALL, call).sendToTarget();  } |

handleMessage 处理消息 MSG\_ADD\_CALL

|  |
| --- |
| /\*\* Default Handler used to consolidate binder method calls onto a single thread. \*/  private final Handler mHandler = new Handler(Looper.getMainLooper()) {  @Override  public void handleMessage(Message msg) {  if (mPhone == null && msg.what != MSG\_SET\_IN\_CALL\_ADAPTER) {  return;  }  switch (msg.what) {  case MSG\_SET\_IN\_CALL\_ADAPTER:  mPhone = new Phone(new InCallAdapter((IInCallAdapter) msg.obj));  onPhoneCreated(mPhone);  break;  case MSG\_ADD\_CALL:  mPhone.internalAddCall((ParcelableCall) msg.obj);  break; |

framworks/base/telecomm/java/android/telecom/Phone.java

作用：一个统一的虚拟设备提供语音手段（和其他）设备上的通信。  
internalAddCall()

|  |
| --- |
| final void internalAddCall(ParcelableCall parcelableCall) {  Call call = new Call(this, parcelableCall.getId(), mInCallAdapter,  parcelableCall.mIsActiveSub);  mCallByTelecomCallId.put(parcelableCall.getId(), call);  mCalls.add(call);  checkCallTree(parcelableCall);  call.internalUpdate(parcelableCall, mCallByTelecomCallId);  fireCallAdded(call);  }  39.1 fireCallAdded()  private void fireCallAdded(Call call) {  for (Listener listener : mListeners) {  listener.onCallAdded(this, call);  }  } |

onCallAdded()是在Listener接口中定义的一个方法，实现它的子类会重写这个方法

|  |
| --- |
| @SystemApi  public final class Phone {  public abstract static class Listener {  ...  public void onCallAdded(Phone phone, Call call) { } |

# 第四部分：CallList–>InCallActivity 开始启动界面 –>显示来电

## 22、onCallAdded 方法在CallList类中的实现

pacakge/apps/InCallUI/src/com/android/incallui/CallList.java

作用：保持主动呼叫的列表和通知感兴趣的类关于这个列表的变化，因为他们是从堆栈收到电话，  
对这个类变化的主要听众是InCallPresenter

|  |
| --- |
| @Override onCallAdded  /\*\*  \* Static singleton accessor method.  \*/  public static CallList getInstance() {  return sInstance;  }  private Phone.Listener mPhoneListener = new Phone.Listener() {  @Override  public void onCallAdded(Phone phone, android.telecom.Call telecommCall) {  Call call = new Call(telecommCall);  if (call.getState() == Call.State.INCOMING) {  onIncoming(call, call.getCannedSmsResponses());  } else {  onUpdate(call);  }  } |

执行了下面的方法onIncoming()

|  |
| --- |
| onIncoming()  /\*\*  \* Called when a single call has changed.  \*/  public void onIncoming(Call call, List textMessages) {  Log.d(this, "onIncoming - " + call);  // Update active subscription from call object. it will be set by  // Telecomm service for incoming call and whenever active sub changes.  if (call.mIsActiveSub) {  long sub = call.getSubId();  Log.d(this, "onIncoming - sub:" + sub + " mSubId:" + mSubId);  if (sub != mSubId) {  setActiveSubscription(sub);  }  }  if (updateCallInMap(call)) {  Log.i(this, "onIncoming - " + call);  }  updateCallTextMap(call, textMessages);  for (Listener listener : mListeners) {  listener.onIncomingCall(call);  }  } |

## 23、界面显示及更新类InCallPresenter

onIncomingCall在InCallpresenter类中的实现

pacakge/apps/InCallUI/src/com/android/incallui/InCallPresenter.java

作用：接受来至CallList的更新并通知InCallActivity（UI）的变化。负责为一个新的呼叫启动活动和当通话断开时结束activity以及Answer、CallCard和CallButton以及VTCall ( MTK加入的VideoCall )的界面更新显示

onIncomingCall是一个接口，以下是它的实现

|  |
| --- |
| /\*\*  \* Called when there is a new incoming call.  \*  \* @param call  \*/  @Override  public void onIncomingCall(Call call) {  InCallState newState = startOrFinishUi(InCallState.INCOMING);//显示来电界面并进行通知  InCallState oldState = mInCallState;  Log.i(this, "Phone switching state: " + oldState + " -> " + newState);  mInCallState = newState;  for (IncomingCallListener listener : mIncomingCallListeners) {  listener.onIncomingCall(oldState, mInCallState, call);  }  if (CallList.getInstance().isDsdaEnabled() && (mInCallActivity != null)) {  mInCallActivity.updateDsdaTab();  }  } |

IncomingCallListener是一个接口，其实现的子类有AnswerPresenter.java, InCallPresenter.java, VideoCallPresenter.java, CallCardPresenter.java, CallButtonPresenter.java, InCallZoomController.java, VideoPauseController.java, SomcCallWidgetPresenter.java, ConferenceManagerPresenter.java。其中每一个Presenter都会有一个相应的Fragment

AnswerPresenter: 负责来电接听/拒接控件的更新显示

InCallPresenter: 负责Answer、CallCard和CallButton以及VTCall ( MTK加入的VideoCall )的界面更新显示

CallCardPresenter:负责联系人信息及通话时间等界面的更新显示

CallButtonPresenter：负责通话界面下方的控制按钮的更新显示

VideoCallPresenter：负责视屏通话的界面更新显示

startOrFinishUi方法中进行当前call状态的判断，更新界面

|  |
| --- |
| private InCallState startOrFinishUi(InCallState newState) {  Log.d(this, "startOrFinishUi: " + mInCallState + " -> " + newState);//INCOMING -> INCALL  // TODO: Consider a proper state machine implementation  boolean isAnyOtherSubActive = InCallState.INCOMING == newState &&  mCallList.isAnyOtherSubActive(mCallList.getActiveSubId());  //If the call is auto answered bring up the InCallActivity  boolean isAutoAnswer = false;  if ((mCallList.getDisconnectedCall() == null) &&  (mCallList.getDisconnectingCall() == null)) {  isAutoAnswer = (mInCallState == InCallState.INCOMING) &&  (newState == InCallState.INCALL) &&  (mInCallActivity == null);  }  Log.d(this, "startOrFinishUi: " + isAutoAnswer);//false  …  showCallUi |= InCallState.PENDING\_OUTGOING == newState && mainUiNotVisible  if (showCallUi || showAccountPicker || isAutoAnswer) {  Log.i(this, "Start in call UI");  showInCall(false /\* showDialpad \*/, !showAccountPicker /\* newOutgoingCall \*/);  } else if (startIncomingCallSequence) {  Log.i(this, "Start Full Screen in call UI");  // We're about the bring up the in-call UI for an incoming call. If we still have  // dialogs up, we need to clear them out before showing incoming screen.  if (isActivityStarted()) {  mInCallActivity.dismissPendingDialogs();  }  if (!startUi(newState)) {  // startUI refused to start the UI. This indicates that it needed to restart the  // activity. When it finally restarts, it will call us back, so we do not actually  // change the state yet (we return mInCallState instead of newState).  return mInCallState;  }  } else if (newState == InCallState.NO\_CALLS) {  // The new state is the no calls state. Tear everything down.  attemptFinishActivity();  attemptCleanup();  }  return newState;  } |

startUI()方法中判断是要启动InCallActivity还是进行通知

|  |
| --- |
| private boolean startUi(InCallState inCallState) {  final Call incomingCall = mCallList.getIncomingCall();  boolean isCallWaiting = mCallList.getActiveCall() != null &&  mCallList.getIncomingCall() != null;  …  boolean anyOtherSubActive = (incomingCall != null &&  mCallList.isAnyOtherSubActive(mCallList.getActiveSubId()));  Log.d(this, "Start UI " + " anyOtherSubActive:" + anyOtherSubActive );//false  if (isCallWaiting || anyOtherSubActive) {  if (mProximitySensor.isScreenReallyOff() && isActivityStarted()) {  Log.i(this, "Restarting InCallActivity to turn screen on for call waiting");  mInCallActivity.finish();  // When the activity actually finishes, we will start it again if there are  // any active calls, so we do not need to start it explicitly here. Note, we  // actually get called back on this function to restart it.  // We return false to indicate that we did not actually start the UI.  return false;  } else {  showInCall(false, false);  }  } else {  mStatusBarNotifier.updateNotification(inCallState, mCallList);//  }  return true;  } |

## 24、updateNotification()更新Notification并进行显示

StatusBarNotifier.java

|  |
| --- |
| public void updateNotification(InCallState state, CallList callList) {  updateInCallNotification(state, callList);  }  private void updateInCallNotification(final InCallState state, CallList callList) {  Log.d(this, "updateInCallNotification...");  final Call call = getCallToShow(callList);  if (call != null) {  showNotification(call);  } else {  cancelNotification();  }  }  private void showNotification(final Call call) {  final boolean isIncoming = (call.getState() == Call.State.INCOMING ||  call.getState() == Call.State.CALL\_WAITING);  if (!TextUtils.isEmpty(mCallId)) {  CallList.getInstance().removeCallUpdateListener(mCallId, this);  }  mCallId = call.getId();  CallList.getInstance().addCallUpdateListener(call.getId(), this);  mContactInfoCache.findInfo(call, isIncoming, new ContactInfoCacheCallback() {  @Override  public void onContactInfoComplete(String callId, ContactCacheEntry entry) {  Call call = CallList.getInstance().getCallById(callId);  if (call != null) {  buildAndSendNotification(call, entry);  }  }  @Override  public void onImageLoadComplete(String callId, ContactCacheEntry entry) {  Call call = CallList.getInstance().getCallById(callId);  if (call != null) {  buildAndSendNotification(call, entry);  }  }  });  }  private void buildAndSendNotification(Call originalCall, ContactCacheEntry contactInfo) {  final Call call = getCallToShow(CallList.getInstance());  if (call == null || !call.getId().equals(originalCall.getId())) {  return;  }  final int state = call.getState();  // Check if data has changed; if nothing is different, don't issue another notification.  final int iconResId = getIconToDisplay(call);  Bitmap largeIcon = getLargeIconToDisplay(contactInfo, call);  String content = getContentString(call);  final String contentTitle = getContentTitle(contactInfo, call);  final boolean isVideoUpgradeRequest = call.getSessionModificationState()  == Call.SessionModificationState.RECEIVED\_UPGRADE\_TO\_VIDEO\_REQUEST;  final int notificationType;  if ((state == Call.State.INCOMING  || state == Call.State.CALL\_WAITING || isVideoUpgradeRequest) &&  !InCallPresenter.getInstance().isShowingInCallUi()) {  notificationType = NOTIFICATION\_INCOMING\_CALL;  } else {  notificationType = NOTIFICATION\_IN\_CALL;  }  if (!checkForChangeAndSaveData(iconResId, content, largeIcon, contentTitle, state,  notificationType)) {  return;  }  if (largeIcon != null) {  largeIcon = getRoundedIcon(largeIcon);  }  …..  if (isVideoUpgradeRequest) {  builder.setUsesChronometer(false);  addDismissUpgradeRequestAction(builder);  addAcceptUpgradeRequestAction(builder);  if (isMoreOptionRequired(call)) {  addMoreAction(builder);  }  } else {  createIncomingCallNotification(call, state, builder);  }  addPersonReference(builder, contactInfo, call);  /\*  \* Fire off the notification  \*/  Notification notification = builder.build();  if (mCurrentNotification != notificationType) {  Log.i(this, "Previous notification already showing - cancelling "  + mCurrentNotification);  mNotificationManager.cancel(mCurrentNotification);  }  Log.i(this, "Displaying notification for " + notificationType);//2  mNotificationManager.notify(notificationType, notification);  mCurrentNotification = notificationType;  }  private void createIncomingCallNotification(  Call call, int state, Notification.Builder builder) {  if (state == Call.State.ACTIVE) {  builder.setUsesChronometer(true);  builder.setWhen(call.getConnectTimeMillis());  } else {  builder.setUsesChronometer(false);  }  // Add hang up option for any active calls (active | onhold), outgoing calls (dialing).  if (state == Call.State.ACTIVE ||  state == Call.State.ONHOLD ||  Call.State.isDialing(state)) {  addHangupAction(builder);  addMuteAction(builder); /\* << [TUL-52] TommySCLin >> \*/  } else if (state == Call.State.INCOMING || state == Call.State.CALL\_WAITING) {  addDismissAction(builder);  if (call.isVideoCall(mContext)) {  addVoiceAction(builder);  if (QtiCallUtils.useExt(mContext)) {  addMoreAction(builder);  } else {  addVideoCallAction(builder);  }  } else {  addAnswerAction(builder);  }  }  } |

showInCall(false /\* showDialpad \*/, !showAccountPicker /\* newOutgoingCall \*/);

showInCall启动InCallActivity

|  |
| --- |
| public void showInCall(final boolean showDialpad, final boolean newOutgoingCall) {  Log.i(this, "Showing InCallActivity");  mContext.startActivity(getInCallIntent(showDialpad, newOutgoingCall));  }  public Intent getInCallIntent(boolean showDialpad, boolean newOutgoingCall) {  final Intent intent = new Intent(Intent.ACTION\_MAIN, null);  intent.setFlags(Intent.FLAG\_ACTIVITY\_NO\_USER\_ACTION | Intent.FLAG\_ACTIVITY\_NEW\_TASK);  intent.setClass(mContext, InCallActivity.class);  if (showDialpad) {  intent.putExtra(InCallActivity.SHOW\_DIALPAD\_EXTRA, true);  }  intent.putExtra(InCallActivity.NEW\_OUTGOING\_CALL\_EXTRA, newOutgoingCall);  return intent;  } |

## 25、SomcAnswerCallViewPad.java接听与挂断按钮界面

|  |
| --- |
| protected void onFinishInflate() {  super.onFinishInflate();  mSlidingTab = (SomcSlidingTab)findViewById(R.id.slidingTab);  mSlidingTab.setOnTriggerListener(new OnTriggerListener() {  public void onTap(View v, int tapState) {  switch (tapState) {  case SomcSlidingTab.OnTriggerListener.LEFT\_HANDLE:  mSlidingTab.setHintText(R.string.phone\_answer\_toast\_txt);  break;  case SomcSlidingTab.OnTriggerListener.RIGHT\_HANDLE:  mSlidingTab.setHintText(R.string.phone\_decline\_toast\_txt);  break;  default:  break;  }  }  //电话接听与挂断界面，左滑接听  public void onTrigger(View v, int whichHandle) {  switch (whichHandle) {  case SomcSlidingTab.OnTriggerListener.LEFT\_HANDLE:  mOnIncomingCallRspListener.onAnswer(VideoProfile.STATE\_AUDIO\_ONLY,  getContext());  break;  case SomcSlidingTab.OnTriggerListener.RIGHT\_HANDLE:  mOnIncomingCallRspListener.onDecline(getContext());  break;  default:  break;  }  }  public void onGrabbedStateChange(View v, int grabbedState) {  }  });  } |

## 26、AnswerPresenter类负责来电接听/拒接控件的更新显示

具体的onAnswer方法在AnswerPresenter.java进行实现

通过调用[acceptUpgradeRequest](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=acceptUpgradeRequest&project=odm-foxconn-android-m-kanuti-tulip-somc)进行通话，在通话前会对通话类型（蓝牙，耳机，普通）进行判断，其具体的实现是在CallAudioManager.java类的setSystemAudioState方法进行实现的。

|  |
| --- |
| [452](http://opengrok.sonymobile.net/JDM-BRANCHES/xref/odm-foxconn-android-m-kanuti-tulip-somc/packages/apps/InCallUI/src/com/android/incallui/AnswerPresenter.java" \l "452) public void [onAnswer](http://opengrok.sonymobile.net/JDM-BRANCHES/s?refs=onAnswer&project=odm-foxconn-android-m-kanuti-tulip-somc)(int [videoState](http://opengrok.sonymobile.net/JDM-BRANCHES/s?refs=videoState&project=odm-foxconn-android-m-kanuti-tulip-somc), [Context](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=Context&project=odm-foxconn-android-m-kanuti-tulip-somc) [context](http://opengrok.sonymobile.net/JDM-BRANCHES/s?refs=context&project=odm-foxconn-android-m-kanuti-tulip-somc)) {  [453](http://opengrok.sonymobile.net/JDM-BRANCHES/xref/odm-foxconn-android-m-kanuti-tulip-somc/packages/apps/InCallUI/src/com/android/incallui/AnswerPresenter.java" \l "453) int [phoneId](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=phoneId&project=odm-foxconn-android-m-kanuti-tulip-somc) = [getActivePhoneId](http://opengrok.sonymobile.net/JDM-BRANCHES/xref/odm-foxconn-android-m-kanuti-tulip-somc/packages/apps/InCallUI/src/com/android/incallui/AnswerPresenter.java#getActivePhoneId)();  [454](http://opengrok.sonymobile.net/JDM-BRANCHES/xref/odm-foxconn-android-m-kanuti-tulip-somc/packages/apps/InCallUI/src/com/android/incallui/AnswerPresenter.java" \l "454) [Log](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=Log&project=odm-foxconn-android-m-kanuti-tulip-somc).i(this, "onAnswer mCallId:" + [mCallId](http://opengrok.sonymobile.net/JDM-BRANCHES/xref/odm-foxconn-android-m-kanuti-tulip-somc/packages/apps/InCallUI/src/com/android/incallui/AnswerPresenter.java#mCallId) + "phoneId:" + [phoneId](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=phoneId&project=odm-foxconn-android-m-kanuti-tulip-somc));  [455](http://opengrok.sonymobile.net/JDM-BRANCHES/xref/odm-foxconn-android-m-kanuti-tulip-somc/packages/apps/InCallUI/src/com/android/incallui/AnswerPresenter.java" \l "455) if ([mCallId](http://opengrok.sonymobile.net/JDM-BRANCHES/xref/odm-foxconn-android-m-kanuti-tulip-somc/packages/apps/InCallUI/src/com/android/incallui/AnswerPresenter.java#mCallId) == [null](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=null&project=odm-foxconn-android-m-kanuti-tulip-somc) || [phoneId](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=phoneId&project=odm-foxconn-android-m-kanuti-tulip-somc) == -1) {  [456](http://opengrok.sonymobile.net/JDM-BRANCHES/xref/odm-foxconn-android-m-kanuti-tulip-somc/packages/apps/InCallUI/src/com/android/incallui/AnswerPresenter.java" \l "456) return;  [457](http://opengrok.sonymobile.net/JDM-BRANCHES/xref/odm-foxconn-android-m-kanuti-tulip-somc/packages/apps/InCallUI/src/com/android/incallui/AnswerPresenter.java" \l "457) }  [458](http://opengrok.sonymobile.net/JDM-BRANCHES/xref/odm-foxconn-android-m-kanuti-tulip-somc/packages/apps/InCallUI/src/com/android/incallui/AnswerPresenter.java" \l "458)  [459](http://opengrok.sonymobile.net/JDM-BRANCHES/xref/odm-foxconn-android-m-kanuti-tulip-somc/packages/apps/InCallUI/src/com/android/incallui/AnswerPresenter.java" \l "459) /\* {{ [TUL-25] TommySCLin \*/  [460](http://opengrok.sonymobile.net/JDM-BRANCHES/xref/odm-foxconn-android-m-kanuti-tulip-somc/packages/apps/InCallUI/src/com/android/incallui/AnswerPresenter.java" \l "460) if ([checkAndShowWaitingCallDialog](http://opengrok.sonymobile.net/JDM-BRANCHES/xref/odm-foxconn-android-m-kanuti-tulip-somc/packages/apps/InCallUI/src/com/android/incallui/AnswerPresenter.java#checkAndShowWaitingCallDialog)()) {  [461](http://opengrok.sonymobile.net/JDM-BRANCHES/xref/odm-foxconn-android-m-kanuti-tulip-somc/packages/apps/InCallUI/src/com/android/incallui/AnswerPresenter.java" \l "461) return;  [462](http://opengrok.sonymobile.net/JDM-BRANCHES/xref/odm-foxconn-android-m-kanuti-tulip-somc/packages/apps/InCallUI/src/com/android/incallui/AnswerPresenter.java" \l "462) }  [463](http://opengrok.sonymobile.net/JDM-BRANCHES/xref/odm-foxconn-android-m-kanuti-tulip-somc/packages/apps/InCallUI/src/com/android/incallui/AnswerPresenter.java" \l "463) /\* }} [TUL-25] \*/  [464](http://opengrok.sonymobile.net/JDM-BRANCHES/xref/odm-foxconn-android-m-kanuti-tulip-somc/packages/apps/InCallUI/src/com/android/incallui/AnswerPresenter.java" \l "464)  [465](http://opengrok.sonymobile.net/JDM-BRANCHES/xref/odm-foxconn-android-m-kanuti-tulip-somc/packages/apps/InCallUI/src/com/android/incallui/AnswerPresenter.java" \l "465) if ([mCall](http://opengrok.sonymobile.net/JDM-BRANCHES/xref/odm-foxconn-android-m-kanuti-tulip-somc/packages/apps/InCallUI/src/com/android/incallui/AnswerPresenter.java#mCall)[[phoneId](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=phoneId&project=odm-foxconn-android-m-kanuti-tulip-somc)].[getSessionModificationState](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=getSessionModificationState&project=odm-foxconn-android-m-kanuti-tulip-somc)()  [466](http://opengrok.sonymobile.net/JDM-BRANCHES/xref/odm-foxconn-android-m-kanuti-tulip-somc/packages/apps/InCallUI/src/com/android/incallui/AnswerPresenter.java" \l "466) == [Call](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=Call&project=odm-foxconn-android-m-kanuti-tulip-somc).[SessionModificationState](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=SessionModificationState&project=odm-foxconn-android-m-kanuti-tulip-somc).[RECEIVED\_UPGRADE\_TO\_VIDEO\_REQUEST](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=RECEIVED_UPGRADE_TO_VIDEO_REQUEST&project=odm-foxconn-android-m-kanuti-tulip-somc)) {  [467](http://opengrok.sonymobile.net/JDM-BRANCHES/xref/odm-foxconn-android-m-kanuti-tulip-somc/packages/apps/InCallUI/src/com/android/incallui/AnswerPresenter.java" \l "467) [Log](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=Log&project=odm-foxconn-android-m-kanuti-tulip-somc).d(this, "onAnswer (upgradeCall) mCallId=" + [mCallId](http://opengrok.sonymobile.net/JDM-BRANCHES/xref/odm-foxconn-android-m-kanuti-tulip-somc/packages/apps/InCallUI/src/com/android/incallui/AnswerPresenter.java#mCallId) + " videoState=" + [videoState](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=videoState&project=odm-foxconn-android-m-kanuti-tulip-somc));  [468](http://opengrok.sonymobile.net/JDM-BRANCHES/xref/odm-foxconn-android-m-kanuti-tulip-somc/packages/apps/InCallUI/src/com/android/incallui/AnswerPresenter.java" \l "468) [InCallPresenter](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=InCallPresenter&project=odm-foxconn-android-m-kanuti-tulip-somc).[getInstance](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=getInstance&project=odm-foxconn-android-m-kanuti-tulip-somc)().[acceptUpgradeRequest](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=acceptUpgradeRequest&project=odm-foxconn-android-m-kanuti-tulip-somc)([videoState](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=videoState&project=odm-foxconn-android-m-kanuti-tulip-somc), [context](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=context&project=odm-foxconn-android-m-kanuti-tulip-somc));  [469](http://opengrok.sonymobile.net/JDM-BRANCHES/xref/odm-foxconn-android-m-kanuti-tulip-somc/packages/apps/InCallUI/src/com/android/incallui/AnswerPresenter.java" \l "469) } else {  [470](http://opengrok.sonymobile.net/JDM-BRANCHES/xref/odm-foxconn-android-m-kanuti-tulip-somc/packages/apps/InCallUI/src/com/android/incallui/AnswerPresenter.java" \l "470) [Log](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=Log&project=odm-foxconn-android-m-kanuti-tulip-somc).d(this, "onAnswer (answerCall) mCallId=" + [mCallId](http://opengrok.sonymobile.net/JDM-BRANCHES/xref/odm-foxconn-android-m-kanuti-tulip-somc/packages/apps/InCallUI/src/com/android/incallui/AnswerPresenter.java#mCallId) + " videoState=" + [videoState](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=videoState&project=odm-foxconn-android-m-kanuti-tulip-somc));  [471](http://opengrok.sonymobile.net/JDM-BRANCHES/xref/odm-foxconn-android-m-kanuti-tulip-somc/packages/apps/InCallUI/src/com/android/incallui/AnswerPresenter.java" \l "471) [TelecomAdapter](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=TelecomAdapter&project=odm-foxconn-android-m-kanuti-tulip-somc).[getInstance](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=getInstance&project=odm-foxconn-android-m-kanuti-tulip-somc)().[answerCall](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=answerCall&project=odm-foxconn-android-m-kanuti-tulip-somc)([mCall](http://opengrok.sonymobile.net/JDM-BRANCHES/xref/odm-foxconn-android-m-kanuti-tulip-somc/packages/apps/InCallUI/src/com/android/incallui/AnswerPresenter.java#mCall)[[phoneId](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=phoneId&project=odm-foxconn-android-m-kanuti-tulip-somc)].[getId](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=getId&project=odm-foxconn-android-m-kanuti-tulip-somc)(), [videoState](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=videoState&project=odm-foxconn-android-m-kanuti-tulip-somc));  [472](http://opengrok.sonymobile.net/JDM-BRANCHES/xref/odm-foxconn-android-m-kanuti-tulip-somc/packages/apps/InCallUI/src/com/android/incallui/AnswerPresenter.java" \l "472) }  [473](http://opengrok.sonymobile.net/JDM-BRANCHES/xref/odm-foxconn-android-m-kanuti-tulip-somc/packages/apps/InCallUI/src/com/android/incallui/AnswerPresenter.java" \l "473)  [474](http://opengrok.sonymobile.net/JDM-BRANCHES/xref/odm-foxconn-android-m-kanuti-tulip-somc/packages/apps/InCallUI/src/com/android/incallui/AnswerPresenter.java" \l "474) [getUi](http://opengrok.sonymobile.net/JDM-BRANCHES/s?defs=getUi&project=odm-foxconn-android-m-kanuti-tulip-somc)().[showIncomingCallView](http://opengrok.sonymobile.net/JDM-BRANCHES/xref/odm-foxconn-android-m-kanuti-tulip-somc/packages/apps/InCallUI/src/com/android/incallui/AnswerPresenter.java#showIncomingCallView)(true); /\* {{ [TUL-25] TommySCLin }} \*/  [475](http://opengrok.sonymobile.net/JDM-BRANCHES/xref/odm-foxconn-android-m-kanuti-tulip-somc/packages/apps/InCallUI/src/com/android/incallui/AnswerPresenter.java" \l "475) }  public void acceptUpgradeRequest(int videoState, Context context) {  Log.d(this, " acceptUpgradeRequest videoState " + videoState);  // Bail if we have been shut down and the call list is null.  if (mCallList == null) {  StatusBarNotifier.clearAllCallNotifications(context);  Log.e(this, " acceptUpgradeRequest mCallList is empty so returning");  return;  }  Call call = mCallList.getVideoUpgradeRequestCall();  if (call != null) {  VideoProfile videoProfile = new VideoProfile(videoState);  call.getVideoCall().sendSessionModifyResponse(videoProfile);  call.setSessionModificationState(Call.SessionModificationState.NO\_REQUEST);  InCallAudioManager.getInstance().onAcceptUpgradeRequest(call, videoState);  }  } |

顺序执行InCallAudioManager.java

|  |
| --- |
| /\*\*  \* Called when user accepts an upgrade request. Route audio to speaker if the user accepts an  \* upgrade request to Video (bidirectional, transmit or receive)  \*/  public void onAcceptUpgradeRequest(final Call call, final int videoState) {  Log.v(this, "onAcceptUpgradeRequest: Call = " + call + "video state = " +  videoState);  if (canEnableSpeaker(call.getVideoState(), videoState)) {  enableSpeaker();  }  }  /\*\*  \* Routes the call to the speaker if audio is not being already routed to Speaker and if  \* bluetooth or wired headset is not connected.  \*/  private static void enableSpeaker() {  final TelecomAdapter telecomAdapter = TelecomAdapter.getInstance();  if (telecomAdapter == null) {  Log.e(LOG\_TAG, "enableSpeaker: TelecomAdapter is null");  return;  }  final int currentAudioMode = AudioModeProvider.getInstance().getAudioMode();  Log.v(LOG\_TAG, "enableSpeaker: Current audio mode is - " + currentAudioMode);  if(QtiCallUtils.isNotEnabled(CallAudioState.ROUTE\_SPEAKER |  CallAudioState.ROUTE\_BLUETOOTH | CallAudioState.ROUTE\_WIRED\_HEADSET,  currentAudioMode)) {  Log.v(LOG\_TAG, "enableSpeaker: Set audio route to speaker");  telecomAdapter.setAudioRoute(CallAudioState.ROUTE\_SPEAKER);  }  } |

顺序执行TelecomAdapter.java

|  |
| --- |
| void setAudioRoute(int route) {  if (mInCallService != null) {  mInCallService.setAudioRoute(route);  } else {  Log.e(this, "error setAudioRoute, mInCallService is null");  }  } |

顺序执行Phone.java

|  |
| --- |
| /\*\*  \* Sets the audio route (speaker, bluetooth, etc...). When this request is honored, there will  \* be change to the {@link #getAudioState()}.  \*  \* @param route The audio route to use.  \*/  public final void setAudioRoute(int route) {  mInCallAdapter.setAudioRoute(route);  } |

顺序执行InCallAdapter.java

|  |
| --- |
| @Override  public void setAudioRoute(int route) {  long token = Binder.clearCallingIdentity();  try {  synchronized (mLock) {  mCallsManager.setAudioRoute(route);  }  } finally {  Binder.restoreCallingIdentity(token);  }  } |

TelecomAdapter类中实现对接听电话类型的判断

顺序执行CallAudioManager.java

|  |
| --- |
| private void setSystemAudioState(boolean isMuted, int route, int supportedRouteMask) {  setSystemAudioState(false /\* force \*/, isMuted, route, supportedRouteMask);  }  private void setSystemAudioState(  boolean force, boolean isMuted, int route, int supportedRouteMask) {  if (!hasFocus()) {  return;  }  CallAudioState oldAudioState = mCallAudioState;  saveAudioState(new CallAudioState(isMuted, route, supportedRouteMask));  if (!force && Objects.equals(oldAudioState, mCallAudioState)) {  return;  }  Log.i(this, "setSystemAudioState: changing from %s to %s", oldAudioState, mCallAudioState);  Log.event(mCallsManager.getForegroundCall(), Log.Events.AUDIO\_ROUTE,  CallAudioState.audioRouteToString(mCallAudioState.getRoute()));  mAudioManagerHandler.obtainMessage(  MSG\_AUDIO\_MANAGER\_SET\_MICROPHONE\_MUTE,  mCallAudioState.isMuted() ? 1 : 0, 0)  .sendToTarget();  // Audio route.  if (mCallAudioState.getRoute() == CallAudioState.ROUTE\_BLUETOOTH) {//蓝牙模式接听  turnOnBluetooth(true);  } else if (mCallAudioState.getRoute() == CallAudioState.ROUTE\_SPEAKER) {//耳机  turnOnBluetooth(false);  turnOnSpeaker(true);  } else if (mCallAudioState.getRoute() == CallAudioState.ROUTE\_EARPIECE ||  mCallAudioState.getRoute() == CallAudioState.ROUTE\_WIRED\_HEADSET) {  turnOnBluetooth(false);  turnOnSpeaker(false);  }  if (!oldAudioState.equals(mCallAudioState)) {  mCallsManager.onCallAudioStateChanged(oldAudioState, mCallAudioState);  updateAudioForForegroundCall();  }  } |

执行完Notification之后再返回InCallPersiter中listener.onIncomingCall

|  |
| --- |
| public void onIncomingCall(Call call) {  InCallState newState = startOrFinishUi(InCallState.INCOMING);  InCallState oldState = mInCallState;  Log.i(this, "Phone switching state: " + oldState + " -> " + newState);  mInCallState = newState;  for (IncomingCallListener listener : mIncomingCallListeners) {  listener.onIncomingCall(oldState, mInCallState, call);  }  if (CallList.getInstance().isDsdaEnabled() && (mInCallActivity != null)) {  mInCallActivity.updateDsdaTab();  }  } |

AnswerPresenter类对listener.onIncomingCall的实现

负责来电接听/拒接控件的更新显示

|  |
| --- |
| public void onIncomingCall(InCallState oldState, InCallState newState, Call call) {  int subId = call.getSubId();  int phoneId = mCalls.getPhoneId(subId);  Log.d(this, "onIncomingCall: " + this);//com.android.incallui.AnswerPresenter@6a9af72  Call modifyCall = mCalls.getVideoUpgradeRequestCall();  if (modifyCall != null) {  showAnswerUi(false);  Log.d(this, "declining upgrade request id: ");  mCalls.removeCallUpdateListener(mCallId[phoneId], this);  InCallPresenter.getInstance().declineUpgradeRequest();  }  if (!call.getId().equals(mCallId[phoneId])) {  // A new call is coming in.  processIncomingCall(call);  }  }  private void processIncomingCall(Call call) {  int subId = call.getSubId();  int phoneId = mCalls.getPhoneId(subId);  mCallId[phoneId] = call.getId();  mCall[phoneId] = call;  mCalls.addCallUpdateListener(mCallId[phoneId], this);//监听当前所有的call的变化  Log.d(TAG, "Showing incoming for call id: " + mCallId[phoneId] + " " + this);  //Call\_0 com.android.incallui.AnswerPresenter@6a9af72  if (showAnswerUi(true)) {  final List<String> textMsgs = mCalls.getTextResponses(call.getId());  configureAnswerTargetsForSms(call, textMsgs);  /\* {{ [TUL-25] TommySCLin \*/  if (call.isVideoCall(getUi().getContext())) {  Call activeCall = CallList.getInstance().getActiveCall();  if (activeCall != null) {  //getUi().showWaitingVideoCallView();  } else {  //getUi().showIncomingVideoCallView();  }  } else {  getUi().showIncomingCallView(CallList.getInstance().getIncomingCall() == null);  }  /\* }} [TUL-25] \*/  }  }  private boolean showAnswerUi(boolean show) {  final InCallActivity activity = InCallPresenter.getInstance().getActivity();  if (activity != null) {  activity.showAnswerFragment(show);  if (getUi() != null) {  getUi().onShowAnswerUi(show);  }  return true;  } else {  return false;  }  } |

在InCallActivity中添加AnswerFragment界面

|  |
| --- |
| public void showAnswerFragment(boolean show) {  showFragment(TAG\_ANSWER\_FRAGMENT, show, true);  }  private void showFragment(String tag, boolean show, boolean executeImmediately) {  Trace.beginSection("showFragment - " + tag);  final FragmentManager fm = getFragmentManagerForTag(tag);  if (fm == null) {  Log.w(TAG, "Fragment manager is null for : " + tag);  return;  }  Fragment fragment = fm.findFragmentByTag(tag);  if (!show && fragment == null) {  // Nothing to show, so bail early.  return;  }  final FragmentTransaction transaction = fm.beginTransaction();  if (show) {  if (fragment == null) {  fragment = createNewFragmentForTag(tag);//创建一个Fragment  transaction.add(getContainerIdForFragment(tag), fragment, tag);  } else {  transaction.show(fragment);  }  } else {  transaction.hide(fragment);  }  transaction.commitAllowingStateLoss();  if (executeImmediately) {  fm.executePendingTransactions();  }  Trace.endSection();  }  private Fragment createNewFragmentForTag(String tag) {  /\* {{ [TUL-23] LichiCLLi \*/  Fragment extFragment = createNewExtFragmentForTag(tag);  if (extFragment != null) {  return extFragment;  }  /\* }} [TUL-23] \*/  if (TAG\_DIALPAD\_FRAGMENT.equals(tag)) {  mDialpadFragment = new DialpadFragment();  return mDialpadFragment;  } else if (TAG\_ANSWER\_FRAGMENT.equals(tag)) {  mAnswerFragment = new AnswerFragment();  return mAnswerFragment;  } else if (TAG\_CONFERENCE\_FRAGMENT.equals(tag)) {  mConferenceManagerFragment = new ConferenceManagerFragment();  return mConferenceManagerFragment;  } else if (TAG\_CALLCARD\_FRAGMENT.equals(tag)) {  mCallCardFragment = new CallCardFragment();  return mCallCardFragment;  }  throw new IllegalStateException("Unexpected fragment: " + tag);  } |

## 27、CallCardPresenter类负责联系人信息及通话时间等界面的更新显示

在CallCardPresenter.java中对于lisrener.incomingCall的实现

在CallCardPresenter类中会对当前正在进行的Call List 进行判断，更新相应的界面信息及电话状态

|  |
| --- |
| @Override  public void onIncomingCall(InCallState oldState, InCallState newState, Call call) {  // same logic should happen as with onStateChange()  onStateChange(oldState, newState, CallList.getInstance());  }  =============================================================================================  @Override  public void onStateChange(InCallState oldState, InCallState newState, CallList callList) {  Log.d(this, "onStateChange() " + newState);  final CallCardUi ui = getUi();  if (ui == null) {  return;  }  Call primary = null;  Call secondary = null;  //创建两个call对象，当当前有一个电话连接，在接收到一个来电时  if (newState == InCallState.INCOMING) {//InCallState最新的状态  primary = callList.getIncomingCall();  } else if (newState == InCallState.PENDING\_OUTGOING || newState == InCallState.OUTGOING) {  primary = callList.getOutgoingCall();  if (primary == null) {  primary = callList.getPendingOutgoingCall();  }  // getCallToDisplay doesn't go through outgoing or incoming calls. It will return the  // highest priority call to display as the secondary call.  secondary = getCallToDisplay(callList, null, true);  } else if (newState == InCallState.INCALL) {  primary = getCallToDisplay(callList, null, false);//显示正在通话的界面  secondary = getCallToDisplay(callList, primary, true);  }  Log.d(this, "Primary call: " + primary);  //[Call\_0, INCOMING, [Capabilities: CAPABILITY\_SUPPORT\_HOLD CAPABILITY\_RESPOND\_VIA\_TEXT CAPABILITY\_MUTE], children:[], parent:null,  //conferenceable:[], videoState:Audio Only, mSessionModificationState:0, VideoSettings:(CameraDir:-1), mIsActivSub:false]  Log.d(this, "Secondary call: " + secondary);//null  final boolean primaryChanged = !(Call.areSame(mPrimary, primary) &&  Call.areSameNumber(mPrimary, primary));  final boolean secondaryChanged = !(Call.areSame(mSecondary, secondary) &&  Call.areSameNumber(mSecondary, secondary));  final boolean shouldShowCallSubject = shouldShowCallSubject(mPrimary);  mSecondary = secondary;  Call previousPrimary = mPrimary;  mPrimary = primary;  if (primaryChanged && shouldShowNoteSentToast(primary)) {  ui.showNoteSentToast();  }  // Refresh primary call information if either:  // 1. Primary call changed.  // 2. The call's ability to manage conference has changed.  if (mPrimary != null && (primaryChanged ||  ui.isManageConferenceVisible() != shouldShowManageConference()) ||  ui.isCallSubjectVisible() != shouldShowCallSubject) {  // primary call has changed  if (previousPrimary != null) {  //clear progess spinner (if any) related to previous primary call  maybeShowProgressSpinner(previousPrimary.getState(),  Call.SessionModificationState.NO\_REQUEST);  CallList.getInstance().removeCallUpdateListener(previousPrimary.getId(), this);  }  CallList.getInstance().addCallUpdateListener(mPrimary.getId(), this);  mPrimaryContactInfo = ContactInfoCache.buildCacheEntryFromCall(mContext, mPrimary,  mPrimary.getState() == Call.State.INCOMING);  updatePrimaryDisplayInfo();  maybeStartSearch(mPrimary, true);  maybeClearSessionModificationState(mPrimary);  }  if (previousPrimary != null && mPrimary == null) {  //clear progess spinner (if any) related to previous primary call  maybeShowProgressSpinner(previousPrimary.getState(),  Call.SessionModificationState.NO\_REQUEST);  CallList.getInstance().removeCallUpdateListener(previousPrimary.getId(), this);  }  if (mSecondary == null) {  // Secondary call may have ended. Update the ui.  mSecondaryContactInfo = null;  updateSecondaryDisplayInfo();  } else if (secondaryChanged) {  // secondary call has changed  mSecondaryContactInfo = ContactInfoCache.buildCacheEntryFromCall(mContext, mSecondary,  mSecondary.getState() == Call.State.INCOMING);  updateSecondaryDisplayInfo();  maybeStartSearch(mSecondary, false);  maybeClearSessionModificationState(mSecondary);  }  /\* {{ [TUL-11] LichiCLLi \*/  if (mPrimary != null && mSecondary != null) {  ui.showMergeButton(mPrimary.can(android.telecom.Call.Details.CAPABILITY\_MERGE\_CONFERENCE) &&  !mPrimary.isVideoCall(mContext) && !mSecondary.isVideoCall(mContext));  }  /\* }} [TUL-11] \*/  /\* {{ [TUL-28] PhilipWYHuang \*/  //Porting from ASD-EDC-FP:19852  if (mPrimary != null) {  ui.showHDVoiceIcon(isHDVoiceCall(mPrimary));  }  /\* }} [TUL-28] \*/  // Start/stop timers.  if (isPrimaryCallActive()) {  Log.d(this, "Starting the calltime timer");  mBaseChronometerTime = mPrimary.getConnectTimeMillis() - System.currentTimeMillis()  + SystemClock.elapsedRealtime();  mCallTimer.start(CALL\_TIME\_UPDATE\_INTERVAL\_MS);  } else {  /\* {{ [TUL-15] LichiCLLi \*/  /\* mark original source code  Log.d(this, "Canceling the calltime timer");  mCallTimer.cancel();  mBaseChronometerTime = 0;  ui.setPrimaryCallElapsedTime(false, 0);  \*/  int timerCallState = (mPrimary == null) ? Call.State.INVALID : mPrimary.getState();  if (isCallTimeVisibleState(timerCallState)) {  // noop.  } else {  Log.d(this, "Canceling the calltime timer");  mCallTimer.cancel();  mBaseChronometerTime = 0;  ui.setPrimaryCallElapsedTime(false, 0);  }  /\* }} [TUL-15] \*/  }  /\* {{ [TUL-15] LichiCLLi \*/  if (mPrimary != null) {  if (mPrimary.getState() != Call.State.DISCONNECTING  && mPrimary.getState() != Call.State.DISCONNECTED) {  ui.showVolteIcon(SomcInCallUiUtils.isImsCall());  }  }  /\* }} [TUL-15] \*/  // Set the call state  int callState = Call.State.IDLE;  if (mPrimary != null) {  callState = mPrimary.getState();  updatePrimaryCallState();  } else {  getUi().setCallState(  callState,  VideoProfile.STATE\_AUDIO\_ONLY,  Call.SessionModificationState.NO\_REQUEST,  new DisconnectCause(DisconnectCause.UNKNOWN),  /\* {{ [TUL-5] KevinKWChung \*/  SomcPreciseDisconnectCause.NOT\_VALID,  /\* }} [TUL-5] \*/  null,  null,  null,  false /\* isWifi \*/,  false /\* isConference \*/);  getUi().showHdAudioIndicator(false);  }  maybeShowManageConferenceCallButton();  // Hide the end call button instantly if we're receiving an incoming call.  getUi().setEndCallButtonEnabled(shouldShowEndCallButton(mPrimary, callState),  callState != Call.State.INCOMING /\* animate \*/);  maybeSendAccessibilityEvent(oldState, newState);  }  private Call getCallToDisplay(CallList callList, Call ignore, boolean skipDisconnected) {  // Active calls come second. An active call always gets precedent.  Call retval = callList.getActiveCall();  if (retval != null && retval != ignore) {  return retval;  }  // Disconnected calls get primary position if there are no active calls  // to let user know quickly what call has disconnected. Disconnected  // calls are very short lived.  if (!skipDisconnected) {  retval = callList.getDisconnectingCall();  if (retval != null && retval != ignore) {  return retval;  }  retval = callList.getDisconnectedCall();  if (retval != null && retval != ignore) {  return retval;  }  }  // Then we go to background call (calls on hold)  retval = callList.getBackgroundCall();  if (retval != null && retval != ignore) {  return retval;  }  // Lastly, we go to a second background call.  retval = callList.getSecondBackgroundCall();  return retval;  } |

MTCall数据库模块

## MTCall信息数据库存储

### 1. 相关类

* CallsManager.java
* CallLogManager.java
* DbModifierWithNotification.java
* CallLogProvider.java

### 2. 过程简述

数据库的存储工作是被CallsManager的回调，将Call信息传给个CallLogManger，电话记录的控制中心，将Call中的数据解析，判断。传入异步任务进行耗时操作，操作为对数据的查询比对，增加，打包，以及最后的数据库插入。

### 3. 调用流程

#### CallsManager.java

private void setCallState(Call call, int newState, String tag)

这个方法在onSuccessfulIncomingCall中调用，简略代码：

private void setCallState(Call call, int newState, String tag) {

if (call == null) {

return;

}

int oldState = call.getState();

Log.i(this, "setCallState %s -> %s, call: %s", CallState.toString(oldState),

CallState.toString(newState), call);

if (newState != oldState) {

call.setState(newState, tag);//改变传入call的状态

Trace.beginSection("onCallStateChanged");

if (mCalls.contains(call)) {

CallsManagerListener somcAmCallsManagerListener = null;

for (CallsManagerListener listener : mListeners) {

if (listener instanceof SomcAmCallsManager) {

somcAmCallsManagerListener = listener;

} else {

listener.onCallStateChanged(call, oldState, newState);//调用回调方法

}

}

if (somcAmCallsManagerListener != null) {

somcAmCallsManagerListener.onCallStateChanged(call, oldState, newState);

}

updateCallsManagerState();

}

}

}

**我们调用了回调方法，回调接口在CallLogManager.java中存在实现关系**

#### CallLogManager.java

public void onCallStateChanged(Call call, int oldState, int newState) {

int disconnectCause = call.getDisconnectCause().getCode();

boolean isNewlyDisconnected =

newState == CallState.DISCONNECTED || newState == CallState.ABORTED;

boolean isCallCanceled = isNewlyDisconnected && disconnectCause == DisconnectCause.CANCELED;

if (isNewlyDisconnected &&

(oldState != CallState.SELECT\_PHONE\_ACCOUNT &&

!call.isConference() &&

!isCallCanceled)) {

int type;

if (!call.isIncoming()) {

type = Calls.OUTGOING\_TYPE;

} else if (disconnectCause == DisconnectCause.MISSED) {

type = Calls.MISSED\_TYPE;

} else {

type = getCallLogTypeForAm(call);//这里获得的type是INCOMING\_TYPE

}

logCall(call, type);//执行此方法

}

}

CallLogManager中的logCall方法主要完成将Call中的信息取出并判断

void logCall(Call call, int callLogType) {

final long creationTime = call.getCreationTimeMillis();

final long age = call.getAgeMillis();

final String logNumber = getLogNumber(call);

final PhoneAccountHandle emergencyAccountHandle =

TelephonyUtil.getDefaultEmergencyPhoneAccount().getAccountHandle();

PhoneAccountHandle accountHandle = call.getTargetPhoneAccount();

if (emergencyAccountHandle.equals(accountHandle)) {

accountHandle = null;

}

SomcAmCallsManager amCallsManager = mCallsManager.getSomcAmCallsManager();

//Answering machine, streams and other properties.

//管理应答机，流管理和其他配置

if (amCallsManager.isThisAnsweringMachineCall(call) && isOkToLogThisCall) {

setAmContentValues(logNumber, call.getHandlePresentation(), creationTime);

}

CallerInfo callerInfo = call.getCallerInfo();

if (callerInfo != null) {

callerInfo.name = SomcTelecomUtils.getNameFromCall(mContext, call);

callerInfo.cnapName = callerInfo.name;

}

logCall(callerInfo, logNumber, call.getHandlePresentation(),

callLogType, callFeatures, accountHandle, creationTime, age, null,

call.isEmergencyCall());//将取出的数据进行下一操作

}

之后执行logCall方法，完成异步任务的准备和开启工作

private void logCall(

CallerInfo callerInfo,

String number,

int presentation,

int callType,

int features,

PhoneAccountHandle accountHandle,

long start,

long duration,

Long dataUsage,

boolean isEmergency) {

final boolean okToLogEmergencyNumber =

mContext.getResources().getBoolean(R.bool.allow\_emergency\_numbers\_in\_call\_log);

final boolean isOkToLogThisCall = !isEmergency || okToLogEmergencyNumber;

sendAddCallBroadcast(callType, duration);//TODO:

if (isOkToLogThisCall) {//数据为true

AddCallArgs args = new AddCallArgs(mContext, callerInfo, number, presentation,

callType, features, accountHandle, start, duration, dataUsage);

logCallAsync(args);//继续执行的方法

if ((callerInfo != null) && (callType != Calls.OUTGOING\_TYPE)

&& (!TextUtils.isEmpty(callerInfo.cnapName))

&& (!TextUtils.isEmpty(number))

&& (!callerInfo.contactExists)) {

new AddCnapTask().execute(args);//true，true，true，true，第五个参数是是否存在于数据库中，不存在则执行

}

}

}

继续执行 logCallAsync(args),新建并开启一个异步任务

public AsyncTask<AddCallArgs, Void, Uri[]> logCallAsync(AddCallArgs args) {

return new LogCallAsyncTask().execute(args);

}

#### LogCallAsyncTask

内部类，根据重写的方法，执行doInBackground，执行结果是我们需要的Uri数组

protected Uri[] doInBackground(AddCallArgs... callList) {

int count = callList.length;

Uri[] result = new Uri[count];//单人或多人通话

for (int i = 0; i < count; i++) {

AddCallArgs c = callList[i];

try {

result[i] = Calls.addCall(c.callerInfo, c.context, c.number, c.presentation,

c.callType, c.features, c.accountHandle, c.timestamp, c.durationInSec,

c.dataUsage, true /\* addForAllUsers \*/);

} catch (Exception e) {

}

}

return result;

}

继续执行Calls.addCall()方法，存在

import android.provider.CallLog.Calls;

因此调用到了CallLog.Calls此方法要完成数据库操作之前的准备工作。包括获取已经存在联系人的Uri，更新DataUsage等操作。

public static Uri addCall(CallerInfo ci, Context context, String number,

int presentation, int callType, int features, PhoneAccountHandle accountHandle,

long start, int duration, Long dataUsage, boolean addForAllUsers, boolean is\_read) {

int numberPresentation = PRESENTATION\_ALLOWED;

TelecomManager tm = null;

try {

tm = TelecomManager.from(context);

} catch (UnsupportedOperationException e) {}

String accountAddress = null;

if (tm != null && accountHandle != null) {

PhoneAccount account = tm.getPhoneAccount(accountHandle);

if (account != null) {

Uri address = account.getSubscriptionAddress();

if (address != null) {

accountAddress = address.getSchemeSpecificPart();//此处简要说明，通过TelecomManager

//获取到本地已经存在的联系人的Uri

}

}

}

/\*正常来电的情况下， 删除的代码块不会被执行\*/

// accountHandle information

String accountComponentString = null;

String accountId = null;

if (accountHandle != null) {

accountComponentString = accountHandle.getComponentName().flattenToString();

accountId = accountHandle.getId();

}

ContentValues values = new ContentValues(6);

//存放信息

values.put(NUMBER, number);

values.put(NUMBER\_PRESENTATION, Integer.valueOf(numberPresentation));

values.put(TYPE, Integer.valueOf(callType));

values.put(FEATURES, features);

values.put(DATE, Long.valueOf(start));

values.put(DURATION, Long.valueOf(duration));

if (dataUsage != null) {

values.put(DATA\_USAGE, dataUsage);

}

values.put(PHONE\_ACCOUNT\_COMPONENT\_NAME, accountComponentString);

values.put(PHONE\_ACCOUNT\_ID, accountId);

values.put(PHONE\_ACCOUNT\_ADDRESS, accountAddress);

values.put(NEW, Integer.valueOf(1));

if (callType == MISSED\_TYPE) {

values.put(IS\_READ, Integer.valueOf(is\_read ? 1 : 0));

}

if ((ci != null) && (ci.contactIdOrZero > 0)) {

final Cursor cursor;

if (ci.normalizedNumber != null) {

final String normalizedPhoneNumber = ci.normalizedNumber;

cursor = resolver.query(Phone.CONTENT\_URI,

new String[] { Phone.\_ID },

Phone.CONTACT\_ID + " =? AND " + Phone.NORMALIZED\_NUMBER + " =?",

new String[] { String.valueOf(ci.contactIdOrZero),

normalizedPhoneNumber},

null);

} else {

}

if (cursor != null) {

try {

if (cursor.getCount() > 0 && cursor.moveToFirst()) {

final String dataId = cursor.getString(0);

updateDataUsageStatForData(resolver, dataId);//将Calls表中的DataUsage字段更新

if (duration >= MIN\_DURATION\_FOR\_NORMALIZED\_NUMBER\_UPDATE\_MS

&& callType == Calls.OUTGOING\_TYPE//条件不成立

&& TextUtils.isEmpty(ci.normalizedNumber)) {

updateNormalizedNumber(context, resolver, dataId, number);//不执行

}

}

} finally {

cursor.close();

}

}

}

Uri result = null;

if (addForAllUsers) {

// Insert the entry for all currently running users, in order to trigger any

// ContentObservers currently set on the call log.

final UserManager userManager = (UserManager) context.getSystemService(

Context.USER\_SERVICE);

List<UserInfo> users = userManager.getUsers(true);

final int currentUserId = userManager.getUserHandle();

final int count = users.size();

for (int i = 0; i < count; i++) {

final UserInfo user = users.get(i);

final UserHandle userHandle = user.getUserHandle();

if (userManager.isUserRunning(userHandle)

&& !userManager.hasUserRestriction(UserManager.DISALLOW\_OUTGOING\_CALLS,

userHandle)

&& !user.isManagedProfile()) {

Uri uri = addEntryAndRemoveExpiredEntries(context,

ContentProvider.maybeAddUserId(CONTENT\_URI, user.id), values);//这里是针对多用户存在的情况下

if (user.id == currentUserId) {

result = uri;

}

}

}

} else {

result = addEntryAndRemoveExpiredEntries(context, CONTENT\_URI, values);//默认的操作，继续执行，得到返回的Uri数组

}

return result;

}

继续执行, 获取ContentProvider去处理插入和一项删除操作，传入的Uri为：

public static final Uri CONTENT\_URI = Uri.parse("content://call\_log/calls");

private static Uri addEntryAndRemoveExpiredEntries(Context context, Uri uri,

ContentValues values) {

final ContentResolver resolver = context.getContentResolver();

Uri result = resolver.insert(uri, values);

resolver.delete(uri, "\_id IN " +

"(SELECT \_id FROM calls ORDER BY " + DEFAULT\_SORT\_ORDER

+ " LIMIT -1 OFFSET 500)", null);

return result;

}

### 数据库操作

根据Uri匹配规则

#### CallLogProvider.java

执行insert方法,在此方法中，进行了线程安全，数据比对，添加其他信息继续执行。

public Uri insert(Uri uri, ContentValues values) {

waitForAccess(mReadAccessLatch);//特殊情况下线程中断等待interrupt方法

checkForSupportedColumns(sCallsProjectionMap, values);//传入的数据是否对应和正确，抛出异常

// Inserting a voicemail record through call\_log requires the voicemail

// permission and also requires the additional voicemail param set.

if (hasVoicemailValue(values)) {

checkIsAllowVoicemailRequest(uri);

mVoicemailPermissions.checkCallerHasWriteAccess(getCallingPackage());

}

if (mCallsInserter == null) {//获取帮助类

SQLiteDatabase db = mDbHelper.getWritableDatabase();

mCallsInserter = new DatabaseUtils.InsertHelper(db, Tables.CALLS);

}

ContentValues copiedValues = new ContentValues(values);

// Add the computed fields to the copied values.

mCallLogInsertionHelper.addComputedValues(copiedValues);//添加了诸如Google位置信息，群组信息等

long rowId = getDatabaseModifier(mCallsInserter).insert(copiedValues);//继续执行

if (rowId > 0) {

return ContentUris.withAppendedId(uri, rowId);

}

return null;

}

调用getDatabaseModifier()将会新建一个DbModifierWithNotification对象。

#### DatabaseModifier.java

private DatabaseModifier getDatabaseModifier(SQLiteDatabase db) {

return new DbModifierWithNotification(Tables.CALLS, db, context());

}

构造方法

public DbModifierWithNotification(String tableName, SQLiteDatabase db, Context context) {

this(tableName, db, null, context);

}

继续调用

private DbModifierWithNotification(String tableName, SQLiteDatabase db,

InsertHelper insertHelper, Context context) {

mTableName = tableName;

mDb = db;

mInsertHelper = insertHelper;

mContext = context;

mBaseUri = mTableName.equals(Tables.VOICEMAIL\_STATUS) ?

Status.CONTENT\_URI : Voicemails.CONTENT\_URI;

mIsCallsTable = mTableName.equals(Tables.CALLS);//进行判断

mVoicemailPermissions = new VoicemailPermissions(mContext);

}

对象新建完成之后要继续进行insert（）方法

public long insert(ContentValues values) {

Set<String> packagesModified = getModifiedPackages(values);

long rowId = mInsertHelper.insert(values);//具体的插入操作细节我们不需要去关心，数据已经完整的赋予了InsertHelper

if (rowId > 0 && packagesModified.size() != 0) {

notifyVoicemailChangeOnInsert(

ContentUris.withAppendedId(mBaseUri, rowId), packagesModified);//通知有语音消息

}

if (rowId > 0 && mIsCallsTable) {

notifyCallLogChange();//执行此方法

}

return rowId;//返回的操作，回头再去看最终返回，对返回的Uri没有操作

}

### 小结

##### 至此，数据库的操作就已经完成

#### 之后便是广播之后的操作，广播之后，会勾起BackupApp的服务，暂时不考虑。