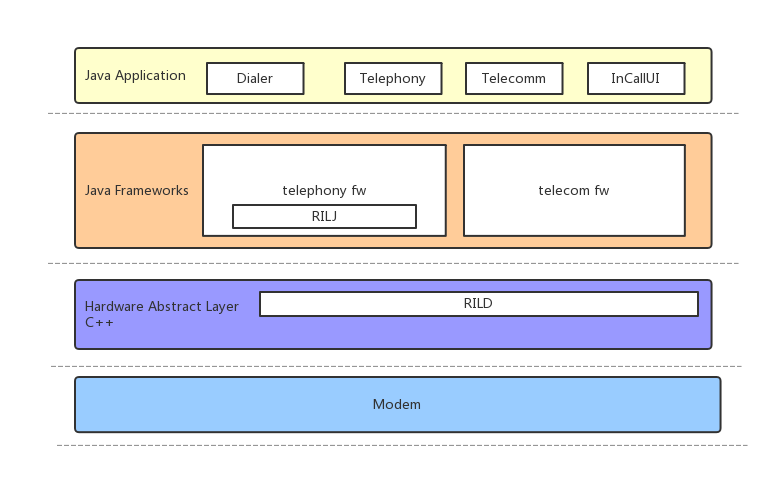
### 框架



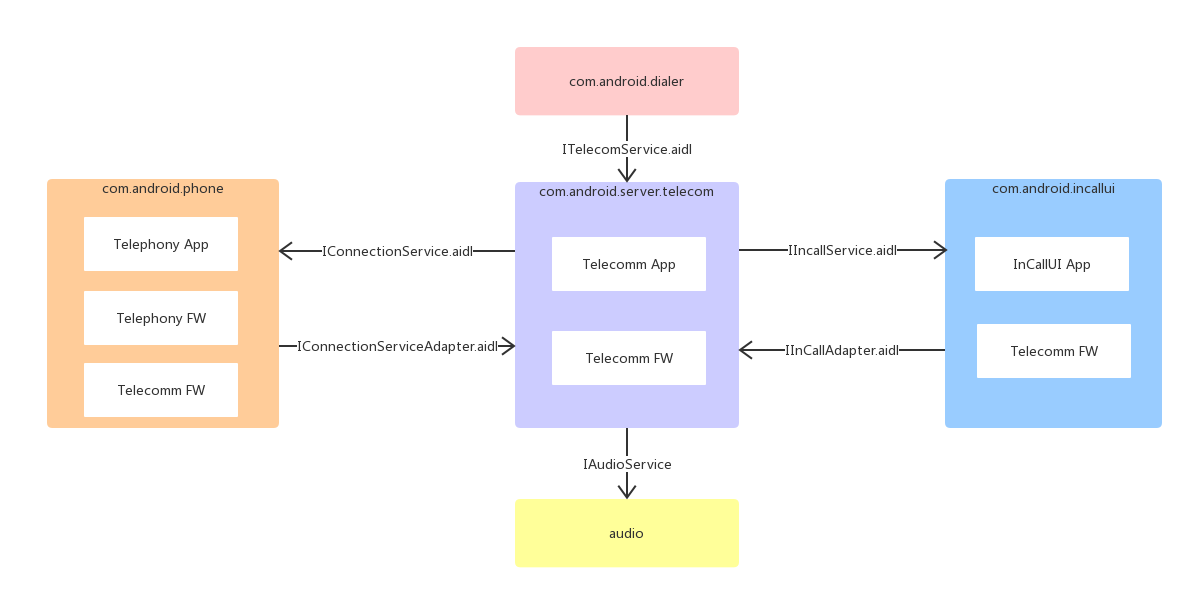
**应用层:**Contacts.apk、TeleService.apk(Phone进程)、Telecom.apk(system\_server进程)、InCallUI.apk

**框架层：**telephony fw(frameworks/opt/telephony)和telecom fw(frameworks/base/telecom)

**RIL：**位于UserLibraries层中的HAL层，提供AP和BP之间的通信功能。

**Modem：**位于BP，负责实际的无线通信。

### 各进程交互的aidl



* ITelecomService.aidl

com.android.server.telecom.TelecomServiceImpl

* IInCallService.aidl

android.telecom.InCallService.InCallServiceBinder(InCallServiceImpl)

* IInCallAdapter.aidl

com.android.server.telecom.InCallAdapter

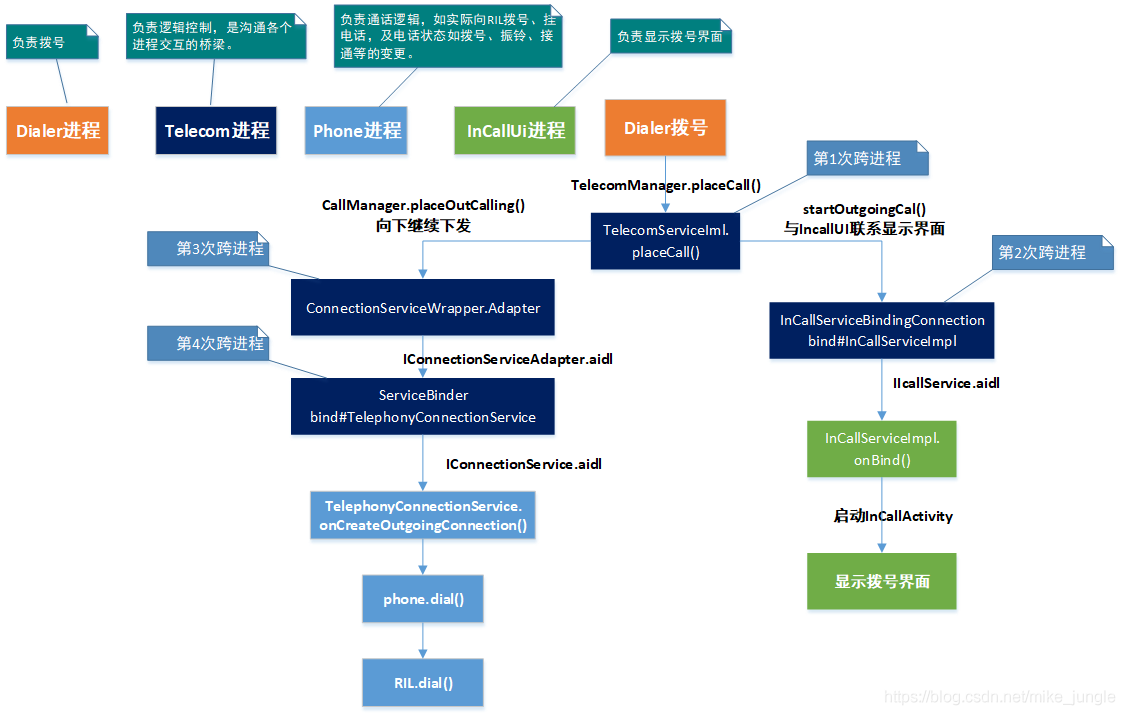
* IConnectionService.aidl

android.telecom.ConnectionService#mBinder

* IConnectionServiceAdapter.aidl

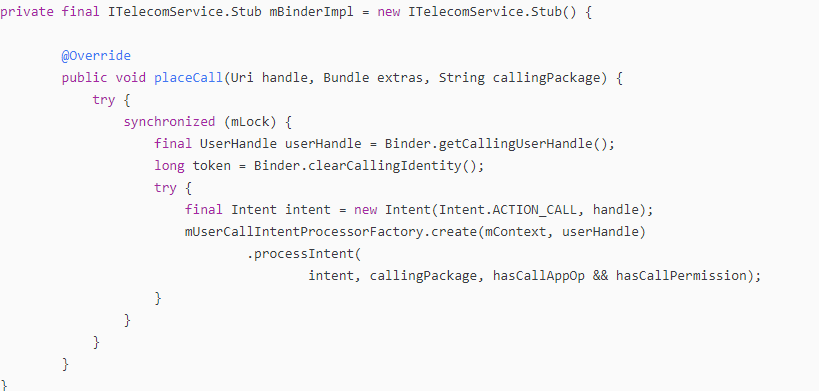
com.android.server.telecom.ConnectionServiceWrapper.Adapter

### 拨号流程

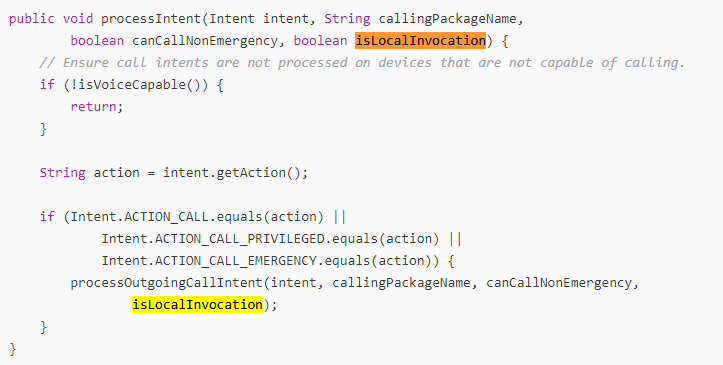


### 代码流程

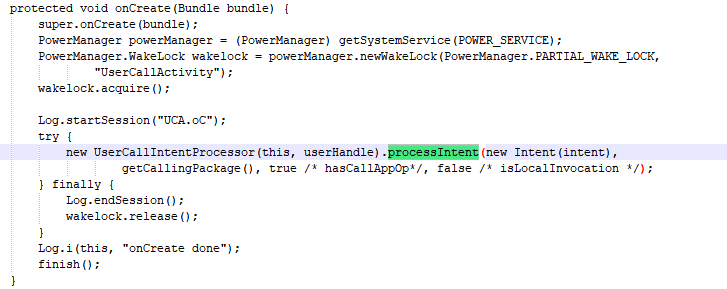
1. 拨号盘拨号
   1. TelecomManager#placeCall



* 1. UserCallIntentProcessor#processIntent

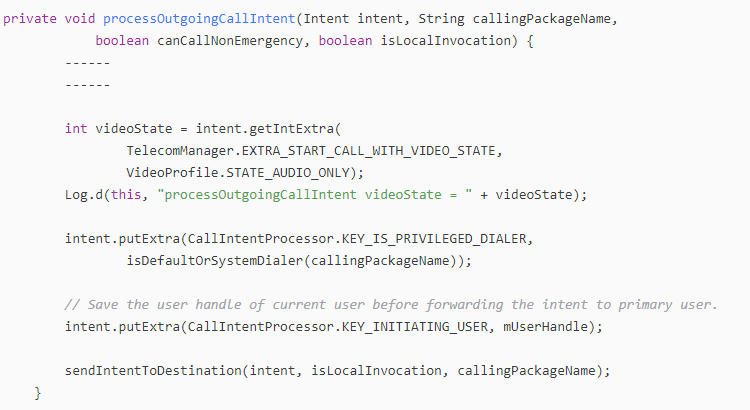


1. Intent拨号
   1. startActivity(new Intent(Intent.ACTION\_CALL).setData(Uri.parse("tel:\*\*\*")));
   2. UserCallActivity#onCreate

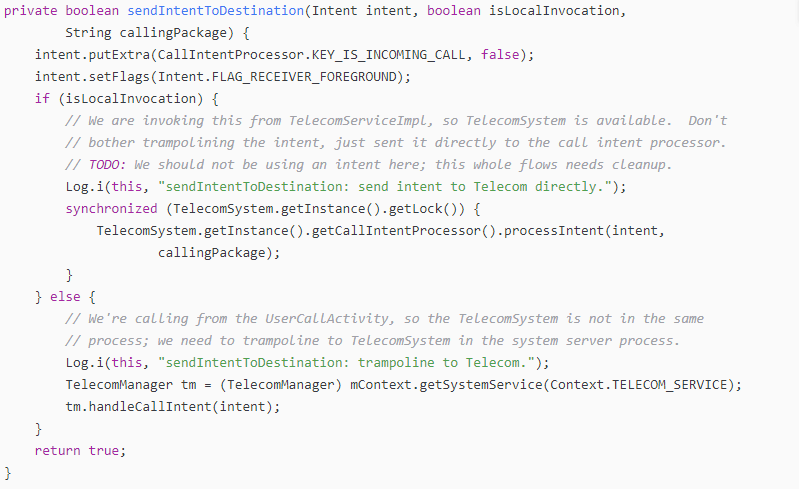


* 1. UserCallIntentProcessor#processIntent

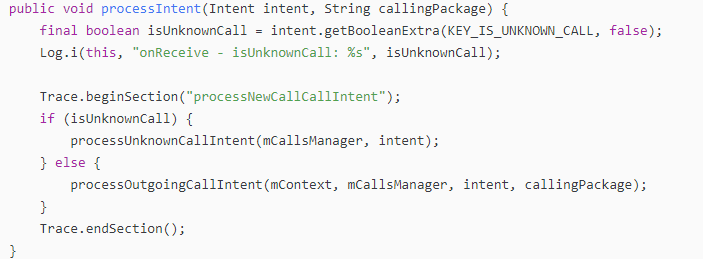
1. UserCallIntentProcessor#processOutgoingCallIntent



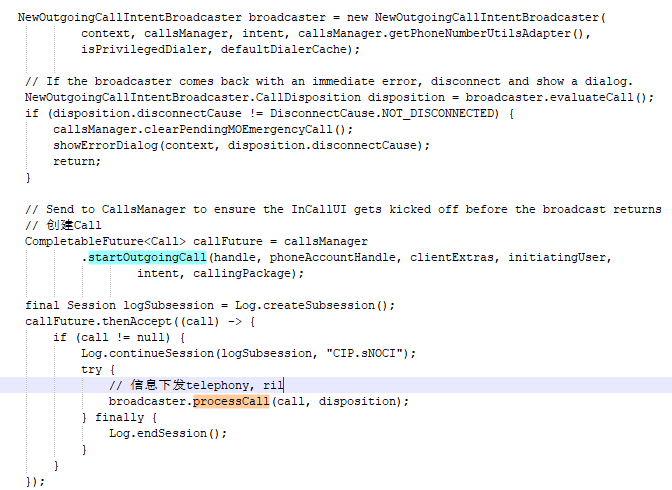
1. UserCallIntentProcessor#sendIntentToDestination



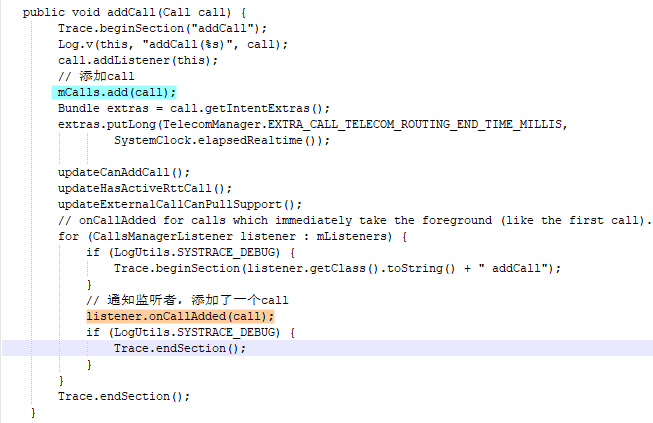
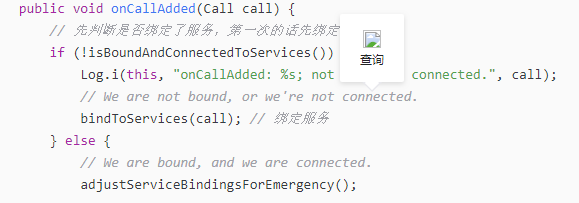
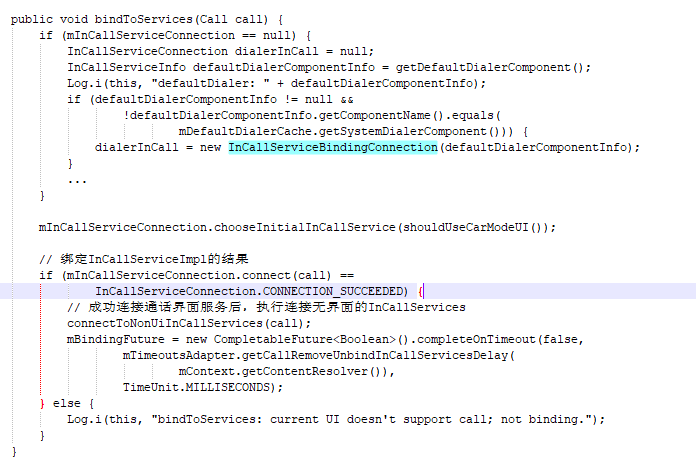
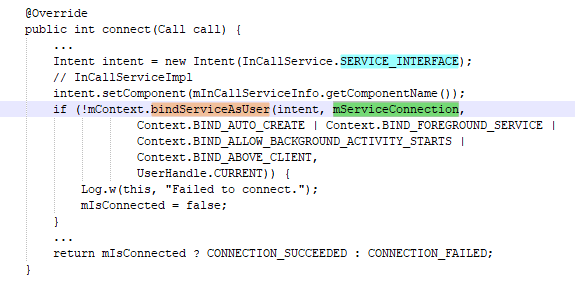
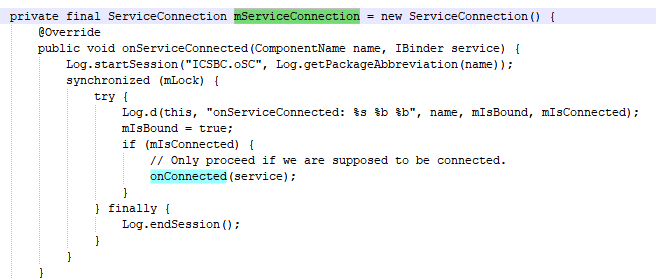
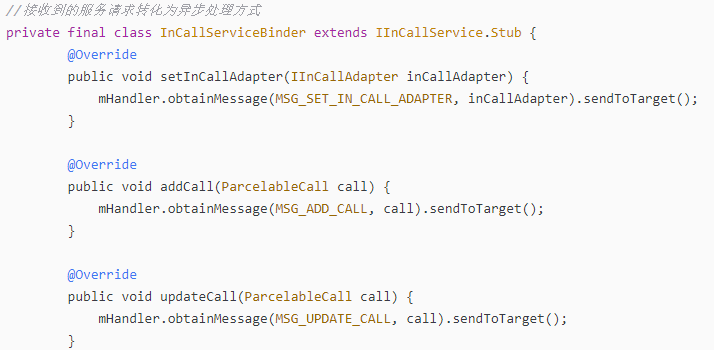
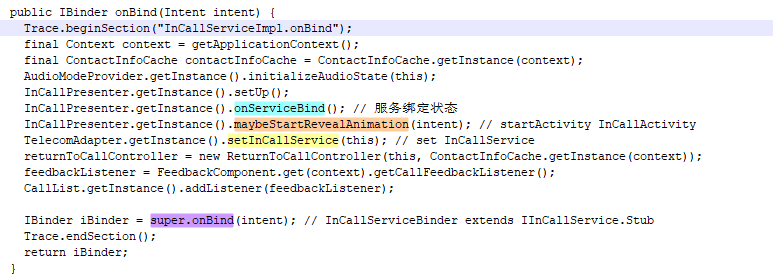
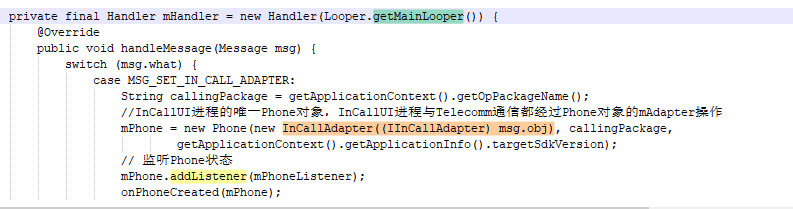
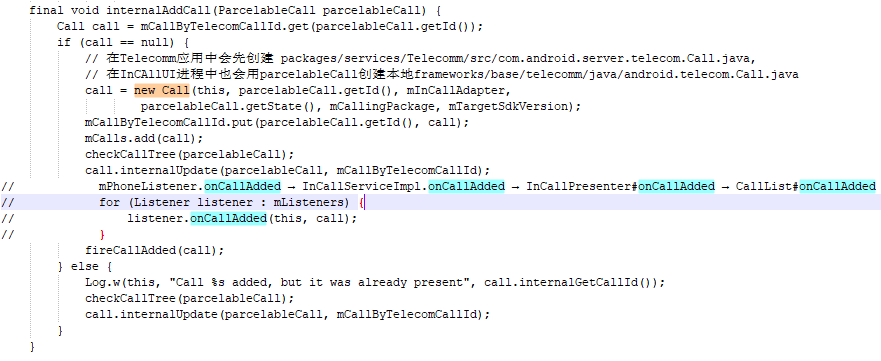
1. CallIntentProcessor#processIntent



1. CallIntentProcessor.Adapter#processOutgoingCallIntent

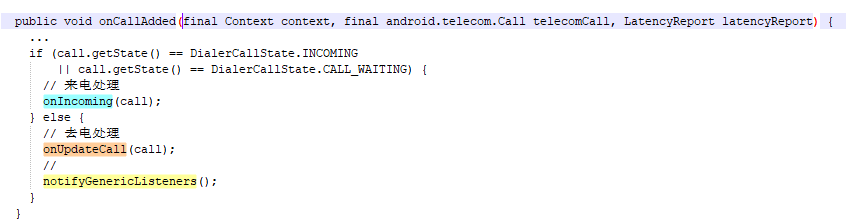


两分支：①与InCallUI通信创建Call建立界面②与Phone通信向协议发起通话请求

1. CallsManager.statOutgoingCall 与InCallUI通信创建、更新、保存Call对象
   1. CallsManager.addCall
   2. InCallController#onCallAdded(用于控制电话App的逻辑和UI)
   3. IncallController#bindToServices建立与InCallUI的跨进程通信，持有InCallServiceImpl的binder代理
   4. InCallController.InCallServiceBindingConnection#connect
   5. InCallController#onConnectedsetInCallAdapter，通过Binder代理对象跨进程访问Telecom应用。
   6. InCallServiceImpl#onBind(InCallServiceImpl InCallService 的实现，负责和 Telecom 沟通更新UI)inCallService.setInCallAdapter
   7. mPhone.internalAddCall 创建一个本地的Telecom Call (不同于system进程的Call, 下面简称TCall)，通知 InCallService
   8. com.android.incallui.call.CallList#onCallAdded

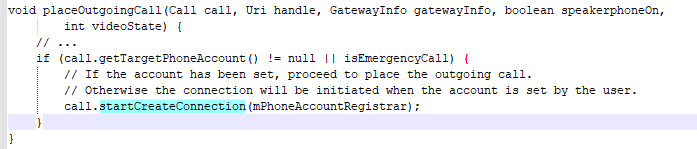
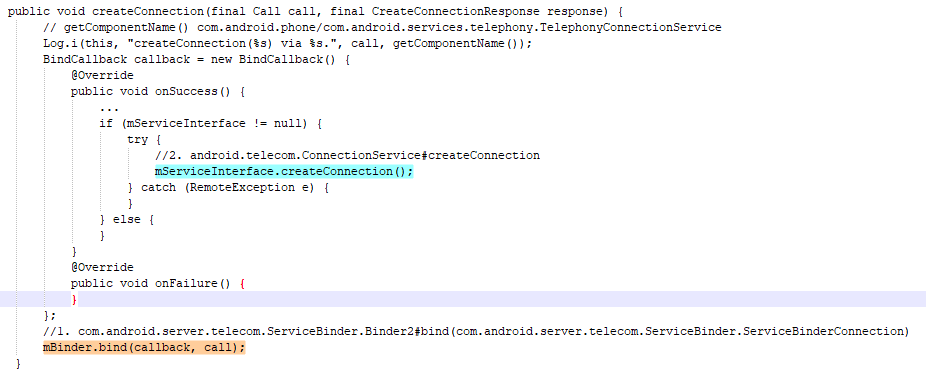
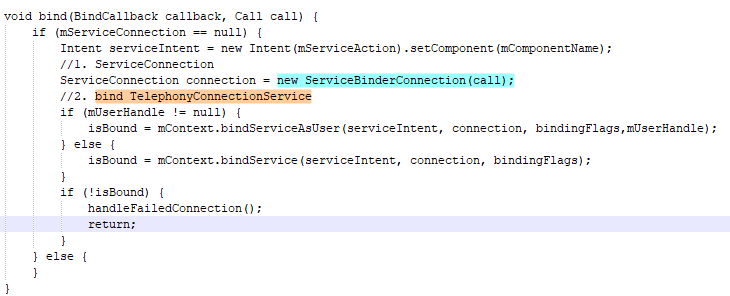
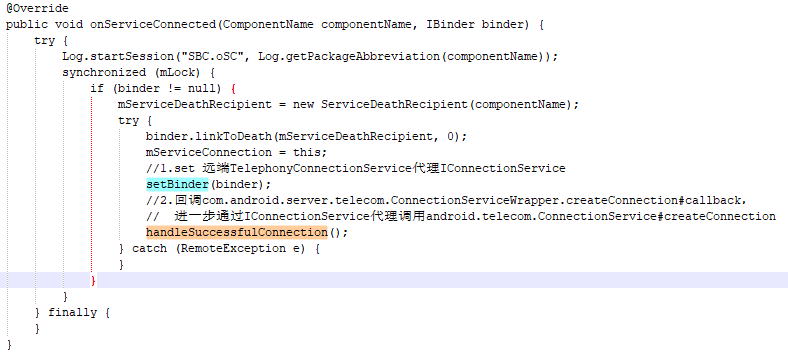
CallList 维护电话列表(CallList里面的Call简称LCall，区别与Phone进程的TCall)

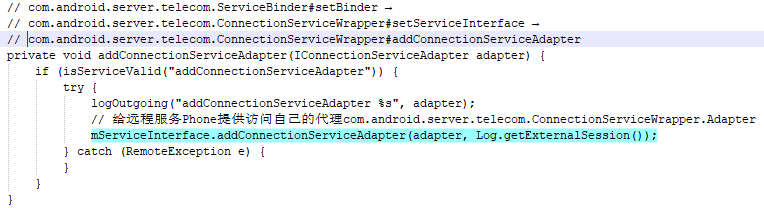
* TCall是由Phone进程维护
* LCall是由InCallUI进程InCallList维护
* TCall和LCAll一一对应

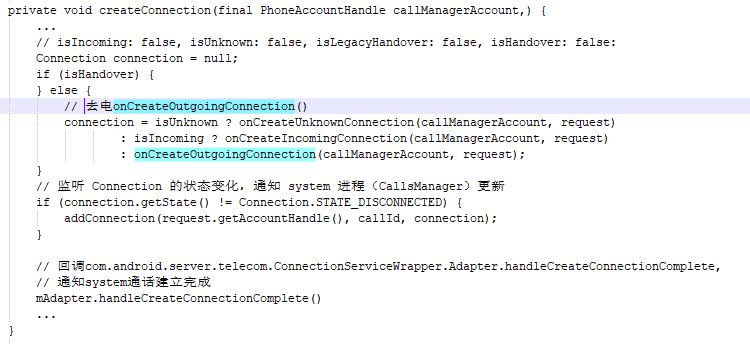
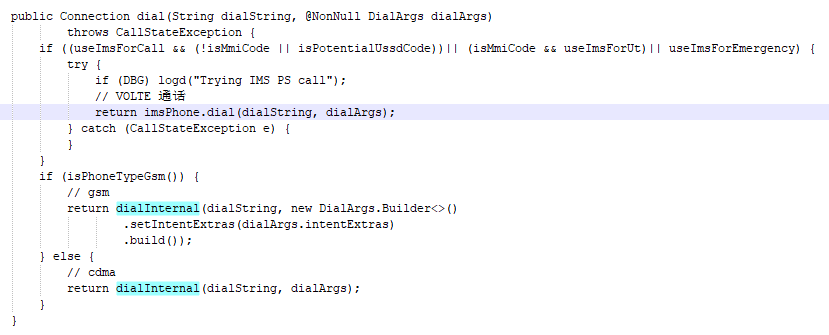
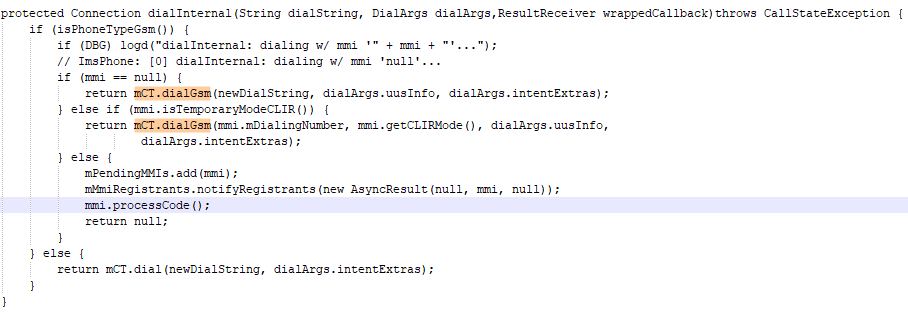
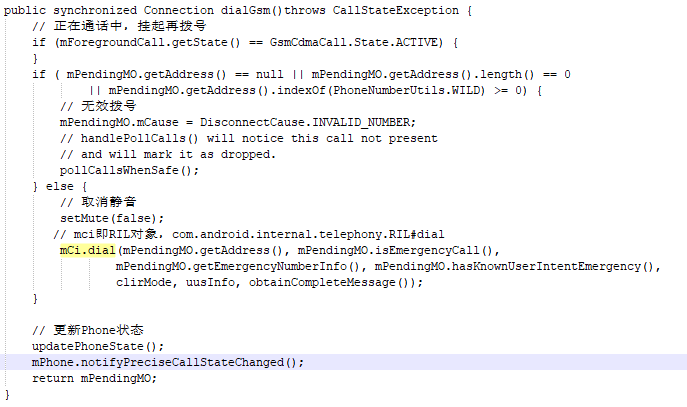
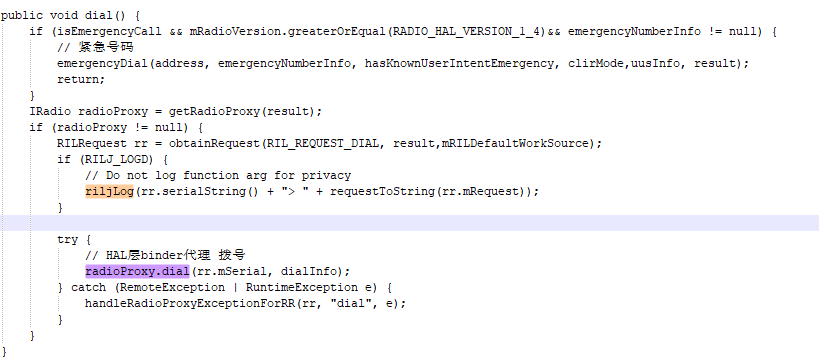


* 1. com.android.incallui.call.CallList#notifyGenericListeners → com.android.incallui.InCallPresenter#onCallListChange

**System下的Telecomm进程的InCallController收到来自CallsManager消息后，通bind InCallService和InCallService.InCallServiceBinder#setInCallAdapter后更新InCallActivity。**

1. NewOutgoingCallIntentBroadcaster#processCall 与Phone通信向Modem发请求，NewOutgoingCallIntentBroadcaster#processCall → NewOutgoingCallIntentBroadcaster#placeOutgoingCallImmediately → CallsManager#placeOutgoingCall
2. Call.startCreateConnection → CreateConnectionProcessor.process → CreateConnectionProcessor.attemptNextPhoneAccount
3. com.android.server.telecom.ConnectionServiceWrapper#createConnection，与com.android.phone/com.android.services.telephony.TelephonyConnectionService连接com.android.server.telecom.ServiceBinder.Binder2#bindcom.android.server.telecom.ServiceBinder.ServiceBinderConnection#onServiceConnected

* **给远程服务Phone提供访问自己的代理**
* **利用远程接口创建一个通话链接**

1. android.telecom.ConnectionService#createConnection //Telecom第二次跨进程访问，绑定服务流程①bindService②addConnectionServiceAdapter③createConnection。createConnection()方法通过判断是来电还是去电分别创建不同的connection，去电则调用onCreateOutgoingConnection()
2. com.android.services.telephony.TelephonyConnectionService#onCreateOutgoingConnection 拦截判断调用TelephonyConnectionService#placeOutgoingConnection()
3. com.android.internal.telephony.GsmCdmaPhone#dial
4. com.android.internal.telephony.GsmCdmaPhone#dialInternal
5. com.android.internal.telephony.GsmCdmaCallTracker#dialGsm
6. com.android.internal.telephony.RIL#dial

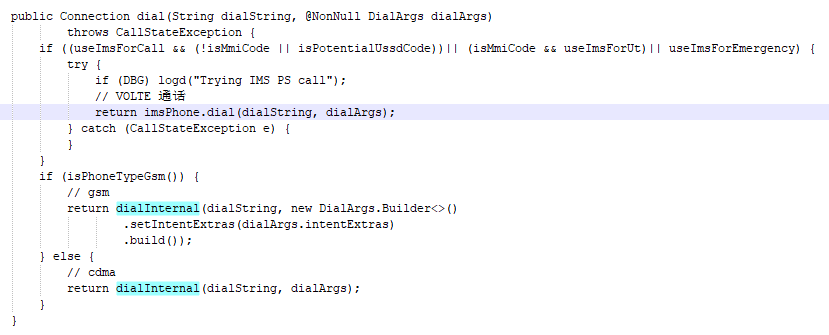
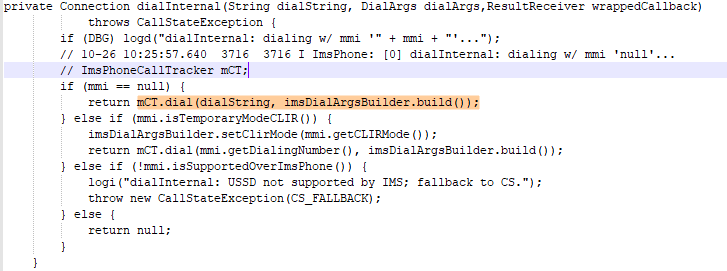
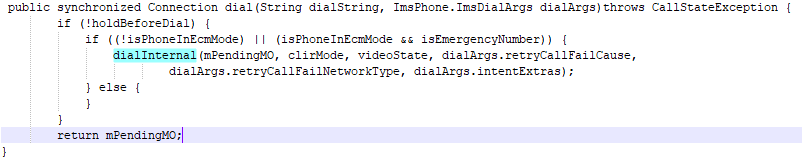
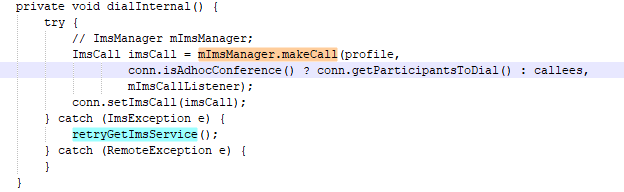
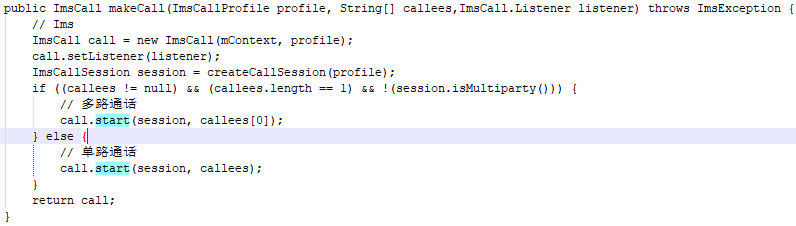
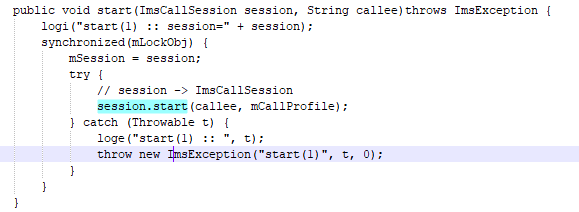
**RIL与通话模块底层通讯过程**

RIL通讯主要由RILSender和RILReceiver构成，用于通讯传输的载体的有 RILRequest,

Registrant(RegistrantList)。

1. RILSender是一个Handler, 通过 #send(RILRequest) 将请求发送到 mSenderThread线程中，handleMessage 将请求写入mSocket中
2. RILReceiver在run内无限轮询，一旦读取到通话底层返回的数据，交给 #processResponse(Parcel)处理。其中应答分为有请求的应答，无请求的应答（即状态改变后的反馈）
   1. RIL\_REQUEST\_xxxx 有请求
   2. RIL\_UNSOL\_xxxx 无请求
3. RILRequest成员变量：
   1. mSerial 请求序号，唯一，保证请求和反馈的一致。
   2. mRequest 请求类型 即 RIL\_xxxx。
   3. mResult 用来发送请求的结果的句柄，由RIL请求的调用者提供。
4. Registrant在 RIL内部维护着一系列的RegistrantList(Registrant的集合)，每个集合代表一种状态改变的类型，对于有请求的应答，RIL是通过对应的mResult来发送结果的，但是对于无请求的应答，RIL是将反馈通知告诉对应的RegistrantList的（#notifyRegistrants），RegistrantList会通知每一个Registrant。在RIL的直接父类定义了这些RegistrantList，并提供了注册到RegistrantList的方法（eg .#registerForCallStateChanged，#unregisterForCallStateChanged）所以，如果想要监听通话状态的变化，那么就需要注册监听到对应的RegistrantList（监听着需要提供一个handler和一个int和object，handler用来发送数据，int区分监听的事件，object是额外的信息）。

补充ImsPhone.dial

1. com.android.internal.telephony.GsmCdmaPhone#dial
2. com.android.internal.telephony.imsphone.ImsPhone#dial → com.android.internal.telephony.imsphone.ImsPhone#dialInternal
3. com.android.internal.telephony.imsphone.ImsPhoneCallTracker#dial()
4. com.android.internal.telephony.imsphone.ImsPhoneCallTracker#dialInternal()
5. com.android.ims.ImsManager#makeCall
6. com.android.ims.ImsCall#start()
7. android.telephony.ims.ImsCallSession#start() → org.codeaurora.ims.ImsCallSessionImpl#start()