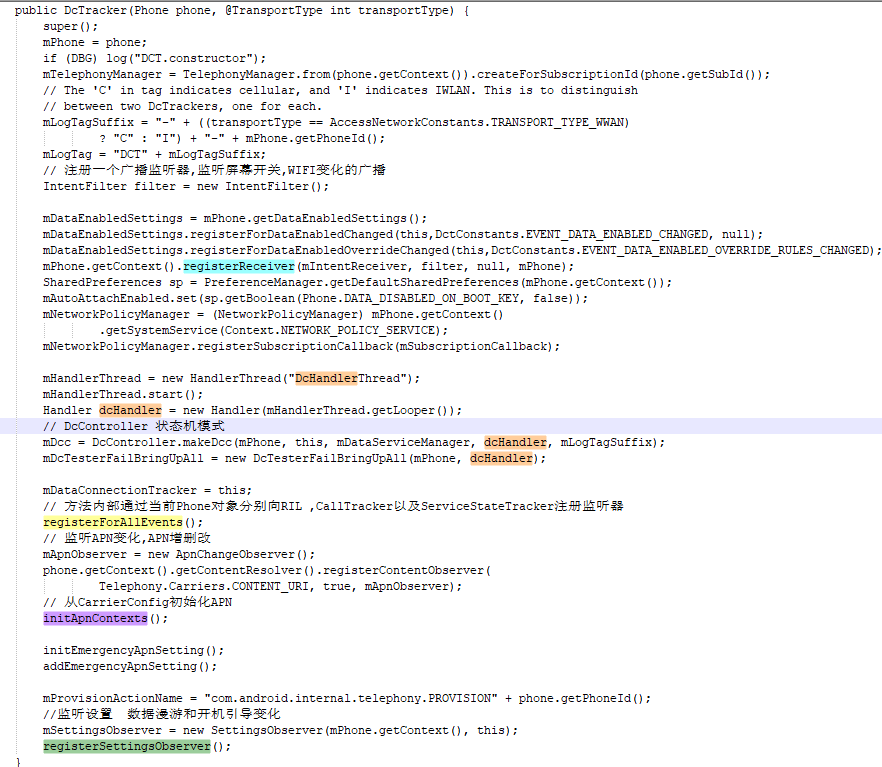
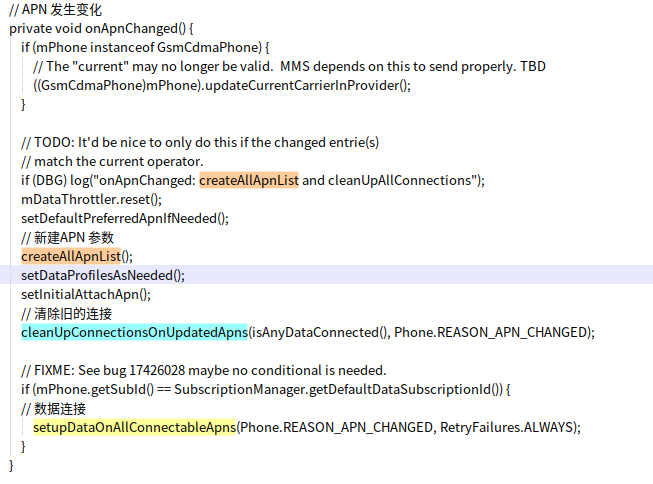
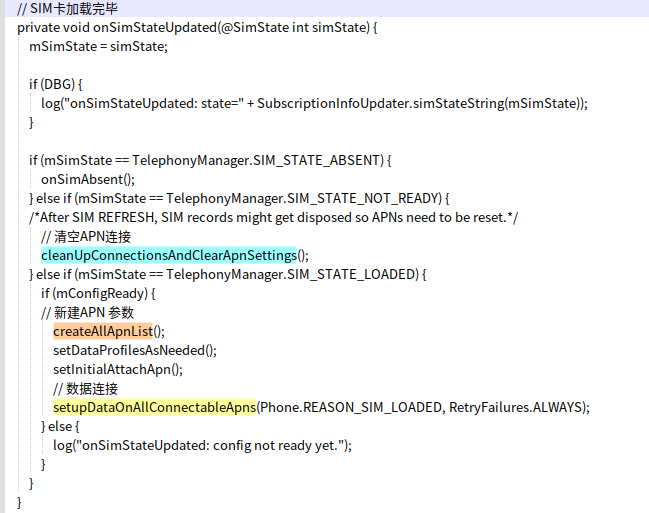
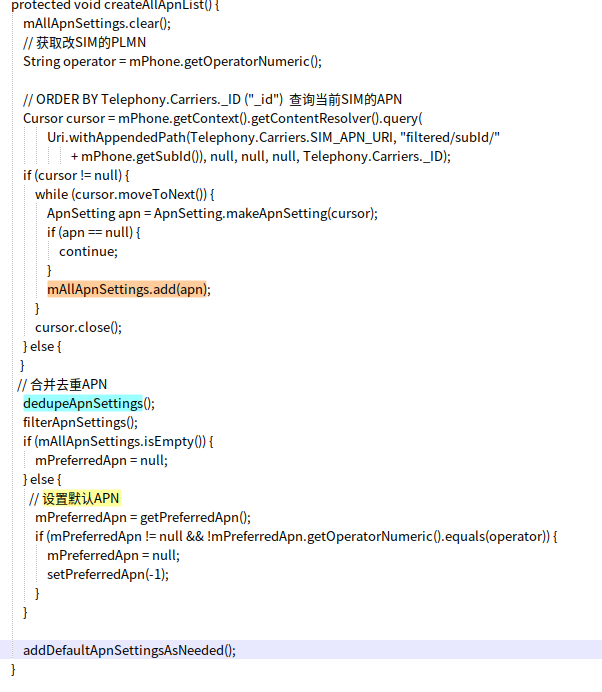
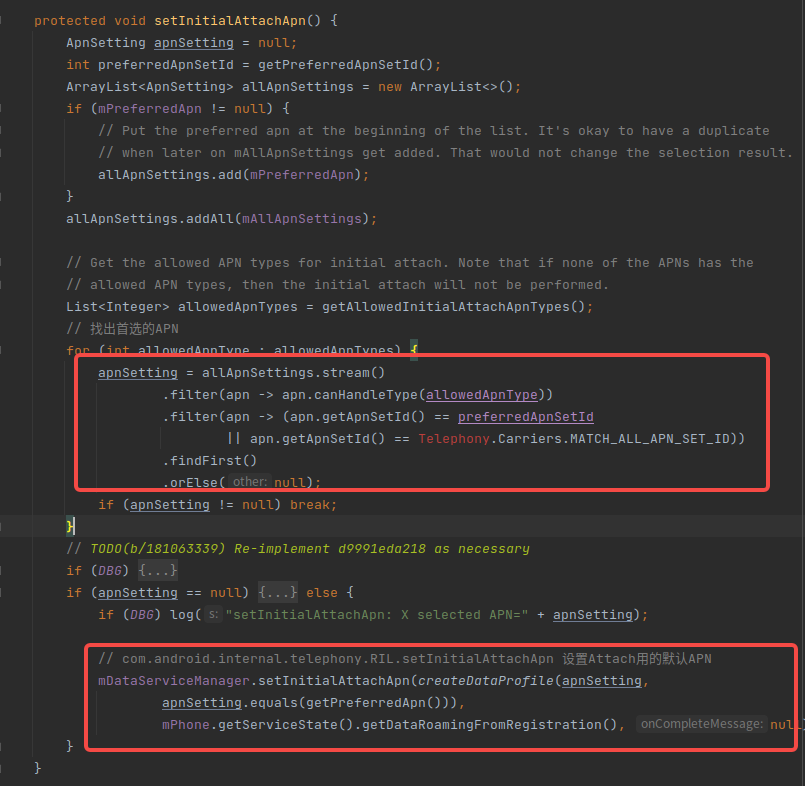
# 数据业务

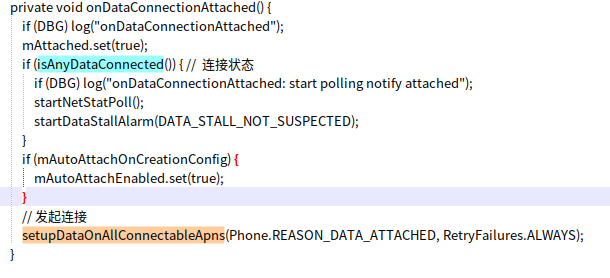
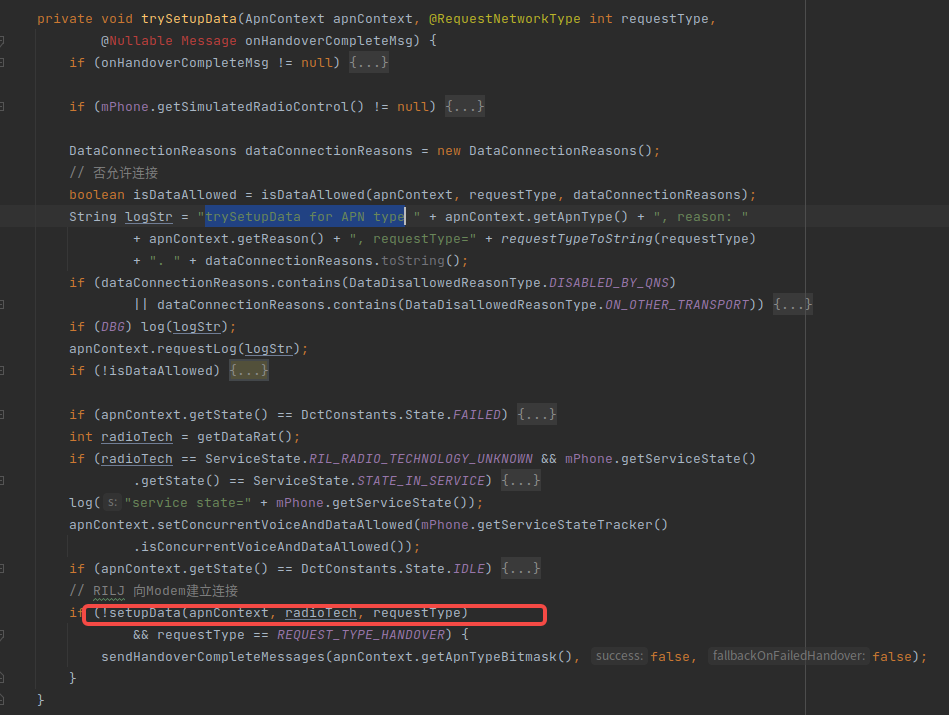
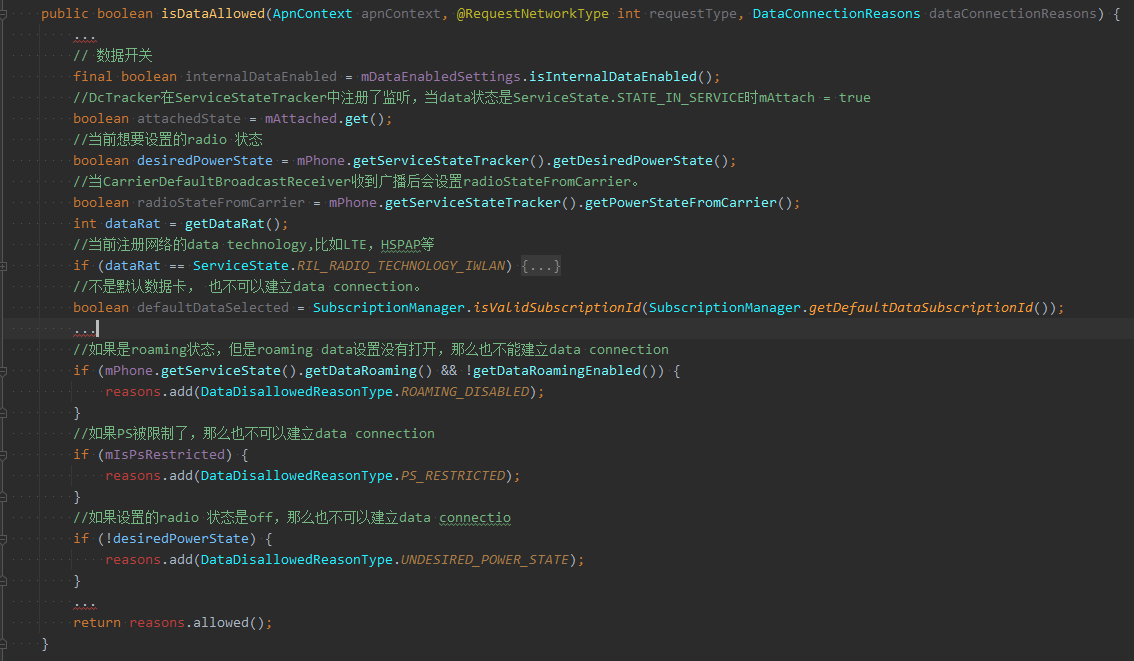
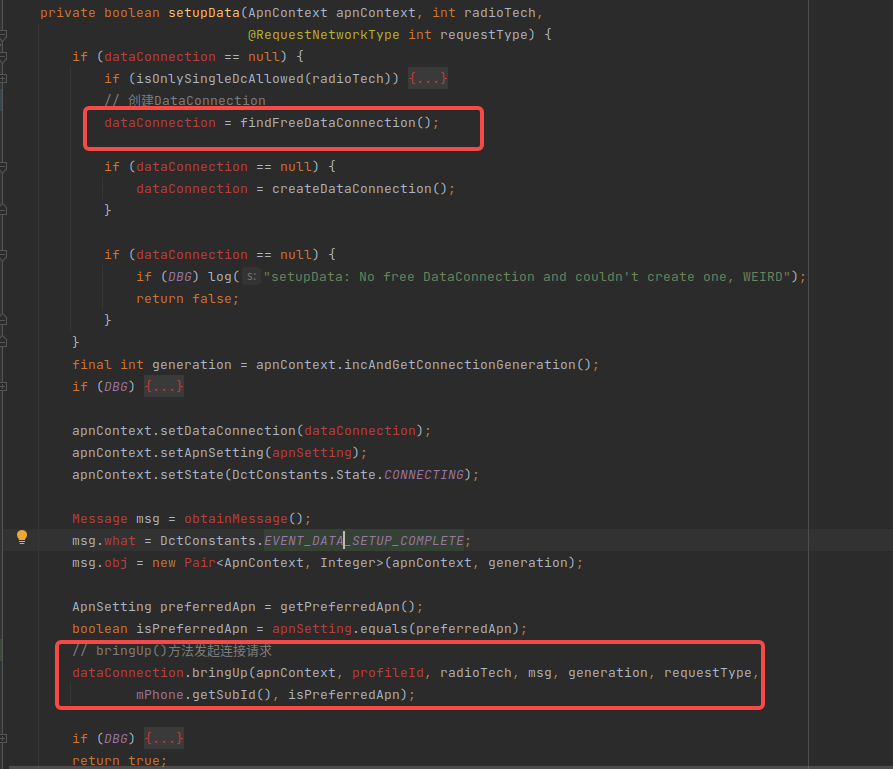
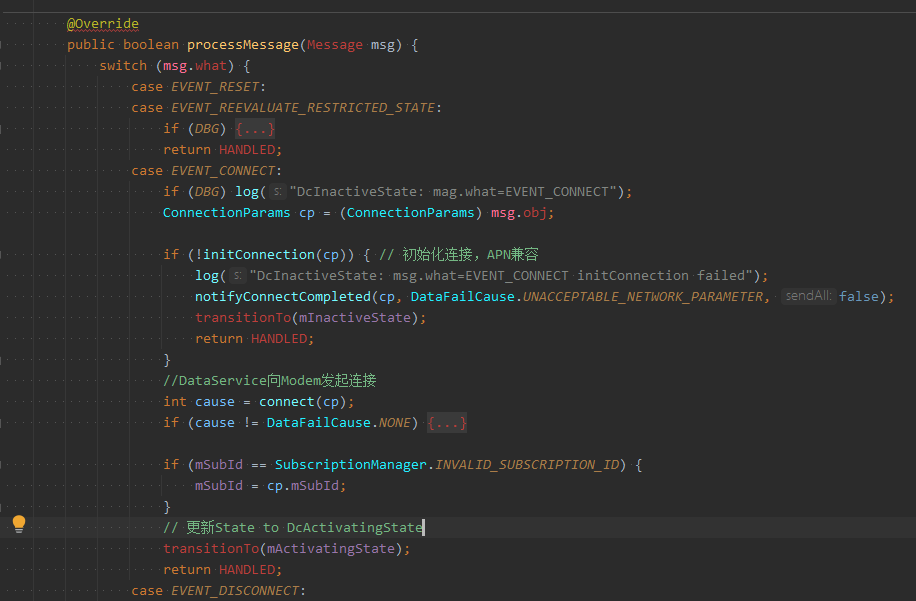
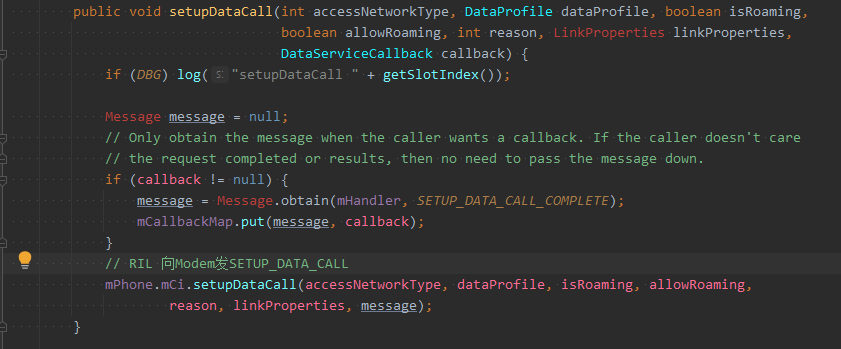
### DcTracker初始化

1. com.android.phone.PhoneApp#onCreate → com.android.phone.PhoneGlobals#onCreate → com.android.internal.telephony.PhoneFactory#makeDefaultPhones → com.android.internal.telephony.PhoneFactory#makeDefaultPhone → com.android.internal.telephony.TelephonyComponentFactory#makePhone → com.android.internal.telephony.GsmCdmaPhone#GsmCdmaPhone() → com.android.internal.telephony.TelephonyComponentFactory#makeDcTracker → com.android.internal.telephony.dataconnection.DcTracker#DcTracker()

### APN创建

1. TelephonyProvider从product/apns-conf.xnl初始化apn数据库
2. 从CarrierConfigLodaer初始化所有APN类型并且添加紧急APN
3. 触发APN创建：SIM卡加载完成，APN发生变化
4. com.android.internal.telephony.dataconnection.DcTracker#createAllApnList 
5. com.android.internal.telephony.dataconnection.DcTracker#setInitialAttachApnAPN准备工作完成，数据连接。
6. com.android.internal.telephony.dataconnection.DcTracker#enableApn APN参数默认都是没有被Enable的， 只有当APN被Enable之后，该APN才可以被使用.

### 数据连接

1. 触发连接：多个事件都会触发连接，如 SIM载入完毕、 PS域Attach成功、 通话结束、 APN改变等，该阶段的最终都是要调用DcTracker#setupDataOnAllConnectableApns()
   1. com.android.internal.telephony.dataconnection.DcTracker#onDataConnectionAttached
2. 准备连接：指在DcTracker收到建立连接的请求之后，需要进行一系列有效性检测，如 APN是否激活、 PS是否就绪、 是否打开网络开关等，然后创建DataConnection()对象，准备发起连接请求
   1. com.android.internal.telephony.dataconnection.DcTracker#setupDataOnAllConnectableApns()
   2. com.android.internal.telephony.dataconnection.DcTracker#trySetupData
   3. com.android.internal.telephony.dataconnection.DcTracker#isDataAllowed
   4. com.android.internal.telephony.dataconnection.DcTracker#setupData
3. 发起连接：指在DataConnection收到DcTracker的请求之后，将请求转交给RILJ的过程，经过该阶段后，请求就发送到了RIL以及Modem层，由底层完成信令的发送和接收
   1. 当消息流进入DataConnection时，网络请求的过程在上层也就进入了最后的发送阶段，接下来我们就来看这最后的阶段如何完成。DataConnection是在刚才的createDataConnection()方法中通过makeDataConnection()方法创建的
   2. com.android.internal.telephony.dataconnection.DataConnection#bringUp → com.android.internal.telephony.dataconnection.DataConnection.DcInactiveState#processMessage(EVENT\_CONNECT) 
   3. com.android.internal.telephony.dataconnection.DataConnection#connect
   4. com.android.internal.telephony.dataconnection.DataServiceManager#setupDataCall → com.android.internal.telephony.dataconnection.CellularDataService.CellularDataServiceProvider#setupDataCall 

### 知识点

1. **PreferedApn**其实就是selected apn，DcTracker和Apnsettings中都会用到。DcTracker在创建data 连接的时候会查询，在连接创建成功后会更新存储;Apnsettings是为用户通过UI操作提供的接口，在显示的时候会查询，当用户切换apn后会更新存储。虽然这两个文件所使用的Uri有差异，但是在TelephonyProvider中这两个Uri会操作同一个sharedpreference文件preferred-apn.xml。
2. **DcController**
   1. DcController内部有一个ArrayList类型的mDcListAll，用于保存所有的DataConnection，每个DataConnection对象都会调用addDc和removeDc将自己添加到mDcListAll或者从mDcListAll删除。
   2. DcController内部维护了一个HashMap: mDcListActiveByCid, 用于保存active的DataConnection; DataConnection会在DcActiveState.enter方法中调用DcController.addActiveDcByCid方法将自己放进mDcListActiveByCid，在DcInactiveState.enter方法中调用DcController.removeActiveDcByCid方法将自己从mDcListActiveByCid中删除。
   3. DcController.DccDefaultState的enter方法在Ril.java内注册了data call list的监听(registerForDataCallListChanged)，当modem上报data call list的变化时，便会调用DccDefaultState.onDataStateChanged方法对新的data call list进行处理。DccDefaultState.onDataStateChanged方法会根据mDcListActiveByCid中保存的DataConnection和传入的DataCallResponse数据对DataConnection进行更新，并通知DcTracker和相应的DataConnection进行相关操作
3. **Update link property：**当Ril收到Data call list变化的消息后，会给DcController发message 即EVENT\_DATA\_STATE\_CHANGED，DcController收到message后会调用onDataStateChanged方法进行处理，进而会调用到DataConnection内的updateLinkProperty方法。ConnectivityService.updateLinkProperties方法会更新MTU、TCP buffer、routes、DNS等链接属性。
4. **MTU的配置**
   1. APN的配置。
   2. modem侧设置。
   3. 通过com.android.internal.R.integer.config\_mobile\_mtu配置。
5. **Data call list**
   1. modem主动上报变化，通过dataCallListChanged方法，相关log: UNSOL\_DATA\_CALL\_LIST\_CHANGED(RIL\_UNSOL\_DATA\_CALL\_LIST\_CHANGED)
   2. AP侧通过getDataCallList方法主动查询，相关log: DATA\_CALL\_LIST(RIL\_REQUEST\_DATA\_CALL\_LIST)