展卓通信

开机 SIM 卡状态

翟少华

2013-01-15

目录

一 开机 RADIO 状态变化

主要内容：RADIO 状态的定义及开机 RADIO 状态的变化

RADIO 状态的定义

Radio 状态定义在 CommandsInterface.java 中，如下：

```
enum RadioState {
    RADIO_OFF,         /* Radio explicitly powered off (eg CFUN=0) */
    RADIO_UNAVAILABLE, /* Radio unavailable (eg, resetting or not booted) */
    RADIO_ON;          /* Radio is on */

    public boolean isOn() /* and available...*/ {
        return this == RADIO_ON;
    }

    public boolean isAvailable() {
        return this != RADIO_UNAVAILABLE;
    }
}
```

其中初始化状态为 RADIO_UNAVAILABLE

RADIO 状态变化时的处理

RILJ 收到主动上报的消息后，进行以下处理

RIL.java

```
case RIL_UNSOL_RESPONSE_RADIO_STATE_CHANGED:
    /* has bonus radio state int */
    RadioState newState = getRadioStateFromInt(p.readInt());  //获取新的 RADIO 状态
    if (RILJ_LOGD) unsljLogMore(response, newState.toString());
    switchToRadioState(newState);  //该函数调用 setRadioState 更新 RADIO 状态
```

BaseCommands.java

```
protected void setRadioState(RadioState newState) {
   RadioState oldState;
```

```
synchronized (mStateMonitor) {
    if (true) {
        Log.v(LOG_TAG, "setRadioState old: " + mState + " new " + newState);
    }

    oldState = mState;
    mState = newState;  //更新状态

    if (oldState == mState) {
        // no state transition
        return;
    }

    mRadioStateChangedRegistrants.notifyRegistrants();
          /*通知 GsmSeriviceStateTracker EVENT_RADIO_STATE_CHANGED，尝试打
          开 radio。 */

    if (mState.isAvailable() && !oldState.isAvailable()) {
        Log.d(LOG_TAG,"Notifying: radio available");
        mAvailRegistrants.notifyRegistrants();// radio available, notifyRegistrants()
        onRadioAvailable();     // radio available
    }

    if (!mState.isAvailable() && oldState.isAvailable()) {
        Log.d(LOG_TAG,"Notifying: radio not available");
                // radio not available, notifyRegistrants()
                mNotAvailRegistrants.notifyRegistrants();
    }

    if (mState.isOn() && !oldState.isOn()) {
        Log.d(LOG_TAG,"Notifying: Radio On");
                // radio on, notifyRegistrants()
        mOnRegistrants.notifyRegistrants();// radio on

    }

    if ((!mState.isOn() || !mState.isAvailable())
        && !((!oldState.isOn() || !oldState.isAvailable()))
    ) {
        Log.d(LOG_TAG,"Notifying: radio off or not available");
        mOffOrNotAvailRegistrants.notifyRegistrants();
    }
}
```

}

RADIO_UNAVAILABLE-----RADIO_OFF

开机 RILJ 收到 UNSOL_RESPONSE_RADIO_STATE_CHANGED RADIO_OFF，原始的 radio 状态为 RADIO_UNAVAILABLE，第一次 RILJ 上报的状态为 RADIO_OFF。所以 radio 状态发生改变，由不可用转为可用，需要通知给所有注册了该变化的类。

注册 radioavailable 的类共三个：GSMPhone，GsmDataConnectionTracker，GsmServiceStateTracker。下面逐一分析。

GSMPhone：从底层获取 BaseBandVersion，IMEI 和 IMEISV。

IMEI：International Mobile Equipment Identification Number，国际移动设备识别码

IMEISV(INTERNATIONAL MOBILE EQUIPMENT IDENTITY SOFTWARE VERSION) SV 表示软件版本。它跟 IMEI 的唯一区别就在于最后一位，IMEI 有 15 位，最后一位是 Check digit，即检验位；

IMEISV 有 16 位，是去掉了 Check digit，加上了两位 SVN，即 software version number。（从 00–98，99 备用）

GsmServiceStateTRacker：直接 break。

GsmDataConnectionTracker：和本次话题无关。

流程图见图 1。

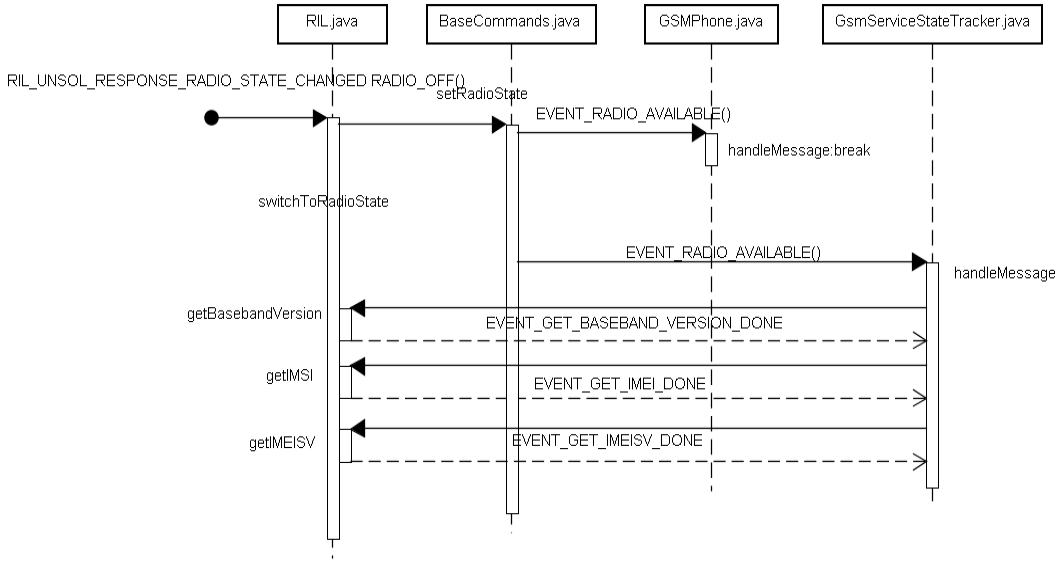


图 1：开机 RADIO 状态变化：RADIO_UNAVAILABLE-RADIO_OFF

RADIO_OFF------- RADIO_ON

radio 开启后，RILJ 收到 UNSOL_RESPONSE_RADIO_STATE_CHANGED RADIO_ON。处理逻辑同上。

radio 状态从 off 到 on，通知给所有注册该变化的类。这里只介绍和 SIM 相关的 GSMPHone 和 IccCard。

GSMPhone 直接 break。

IccCard：查询 IccCardStatus。根据返回结果更新 sim 相关状态，如果 SIM 卡状态变化，通知相关类。具体详见 SIM 卡状态变化。

到这里，radio 相关的就基本结束了。可以看出 radio 的初始状态为 RADIO_UNAVAILABLE，radio_off 的消息是 RIL 主动上报的，那么 radio 状态是怎么变为 radio_on 的呢。开启 radio 是上层下发的命令呢还是底层主动开启的呢？

二 开机 SIM 状态变化

SIM 卡状态定义

SIM 卡状态 IccCard.java

```
public enum State {
        UNKNOWN,          // 未知状态
        ABSENT,           // 没有 SIM 卡插入
        PIN_REQUIRED,     // 需要输入 PIN 码
        PUK_REQUIRED,     // PIN 码解锁失败，需要输入 PUK 码
        NETWORK_LOCKED,   // 锁网
        READY,            // 就绪
        BLOCKED,          // PUK 解锁失败
        NOT_READY,        // 未就绪
        PERM_DISABLED;    // ？？？？？？？？？？？？？？

    public boolean isPinLocked() {
        return ((this == PIN_REQUIRED) || (this == PUK_REQUIRED));
    }
```

```
    public boolean iccCardExist() {
        return ((this == PIN_REQUIRED) || (this == PUK_REQUIRED)
                || (this == NETWORK_LOCKED) || (this == READY)
                || (this == PERM_DISABLED));
    }
}
```

SIM 卡状态变化

RILJ 上报 RIL_UNSOL_RESPONSE_SIM_STATUS_CHANGED

IccCard 注册监听了 SIM 卡状态变化 mPhone.mCM.registerForIccStatusChanged (mHandler,EVENT_ICC_STATUS_CHANGED, null);

RIL 上报 SIM 状态变化的消息时，会收到消息 EVENT_ICC_STATUS_CHANGED，收到该消息后 IccCard 向底层发送查询卡状态请求。

```
case EVENT_ICC_STATUS_CHANGED:
Log.d(mLogTag, "Received Event EVENT_ICC_STATUS_CHANGED");
mPhone.mCM.getIccCardStatus(obtainMessage(EVENT_GET_ICC_STATUS_DONE));
break;
```

请求返回后调用 handleIccCardStatus 进一步处理。根据卡状态的不同分以下几种情况：

transitionedIntoCardPresent：IccCard:newState = NOT_READY oldState = null。

发送 intent SIM_CARD_PRESENT。GsmServiceStateTracker 收到该 intent 后获取消息 EVENT_ICC_STATUS_CHANGED，处理该消息时会向底层发送请求开启 radio，并通过 pollstate 获取运营商，注网状态等信息，但是此时 radio 处于非开启状态，所以不会向底层发送查询命令，但是会进行服务状态，如信号强度等的初始化工作。

transitionedIntoIccReady：IccCard:newState = READY oldState = NOT_READY。

发送 intent ACTION_GET_ICC_STATUS_DONE，但是代码中没有找到接收该 intent 的 receiver。

通知注册了该变化的类：GsmServiceStateTracker，收到消息 EVENT_SIM_READY，处理该消息时进一步注册监听 simloaded 消息，并通过 pollState 方法更新服务状态等。这时 radio 状态应该处于 on 的状态，所以会向底层发送查询 gprs 状态，服务状态等请求。

另外，通过 mIccRecords.onReady() 的调用开始读取 SIM 文件，读取完成后发送 simloaded 广播。

TransitionedIntoPinLocked ： IccCard:newState = PIN_REQUIRED oldState = NOT_READY。

发送广播 INTENT_VALUE_ICC_LOCKED，跟新 statusbar 相关的 sim 状态和 keyguard 相关的 sim 状态等

通知注册监听了该变化的 IccCard，向底层发请求查询 CB_FACILITY_BA_SIM，即查

询 PIN 码。

另外还有 transitionedIntoAbsent 等多种变化，不再一一介绍，详见 IccCard.java。
以下是几种常见情况下的 sim 卡状态变化：

是正常开机状态变化：
IccCard:newState = NOT_READY oldState = null
IccCard:newState = READY oldState = NOT_READY
IccCard:newState = READY oldState = READY

开启飞行模式开机：
IccCard:newState = NOT_READY oldState = null
然后关闭飞行模式：
IccCard:newState = READY oldState = NOT_READY
IccCard:newState = READY oldState = READY

开启 pin 码开机
IccCard:newState = NOT_READY oldState = null
IccCard:newState = PIN_REQUIRED oldState = NOT_READY
IccCard:newState = PIN_REQUIRED oldState = PIN_REQUIRED
IccCard:newState = READY oldState = PIN_REQUIRED
IccCard:newState = READY oldState = READY

无 sim 卡开机：
IccCard:newState = ABSENT oldState = null

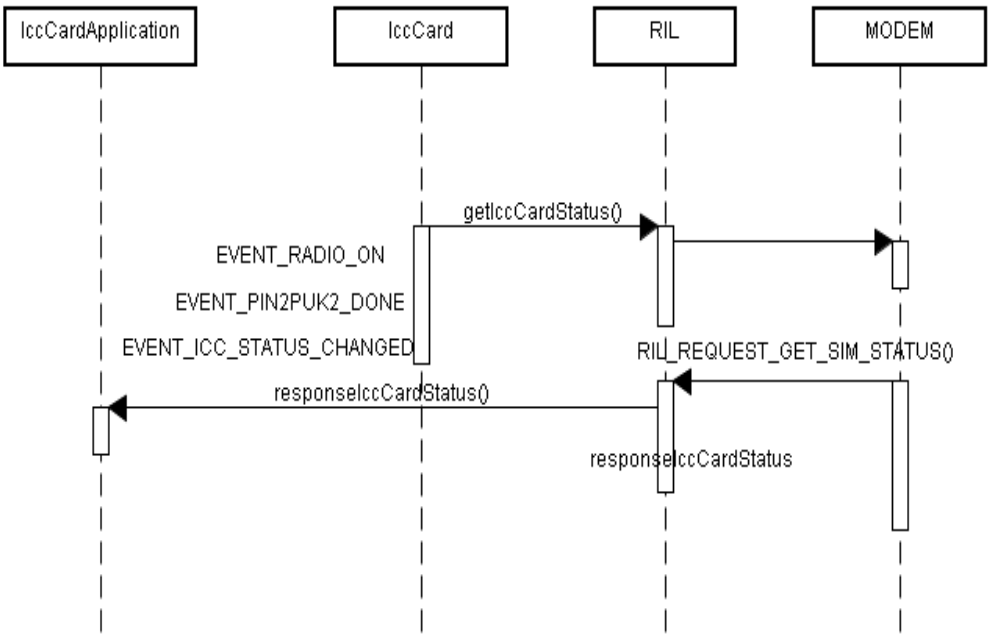
SIM 卡状态的跟新

上文中的 newState 是通过 getIccCardState 方法获得的，具体如下：

```
public State getIccCardState() {
      if(!is3gpp && !isSubscriptionFromIccCard) {
            // CDMA can get subscription from NV. In that case,
            // subscription is ready as soon as Radio is ON.
          return State.READY;
      }

      if (mIccCardStatus == null) {
          Log.e(mLogTag, "[IccCard] IccCardStatus is null");
          return IccCard.State.ABSENT;
      }
```

```
        // this is common for all radio technologies
        if (!mIccCardStatus.getCardState().isCardPresent()) {
            return IccCard.State.ABSENT;
        }

        RadioState currentRadioState = mPhone.mCM.getRadioState();
        // check radio technology
        if( currentRadioState == RadioState.RADIO_OFF         ||
            currentRadioState == RadioState.RADIO_UNAVAILABLE) {
            return IccCard.State.NOT_READY;
        }

        if( currentRadioState == RadioState.RADIO_ON ) {
            State csimState =
                getAppState(mIccCardStatus.getCdmaSubscriptionAppInde
x());
            State usimState =
                getAppState(mIccCardStatus.getGsmUmtsSubscriptionAppI
ndex());

            if(mDbg) log("USIM=" + usimState + " CSIM=" + csimState);

            if (mPhone.getLteOnCdmaMode() == Phone.LTE_ON_CDMA_TRUE)
{
                // UICC card contains both USIM and CSIM
                // Return consolidated status
                return getConsolidatedState(csimState, usimState,
csimState);
            }

            // check for CDMA radio technology
            if (!is3gpp) {
                return csimState;
            }
            return usimState;
        }

        return IccCard.State.ABSENT;
    }
```

该方法也是通过AppState来判断返回状态的。app_state是根据RIL上报的状态更新的，具体流程如下：



SIM 卡文件读取

SIM 卡状态变为 ready 后， SIMRecords.java 会通过 fetchSimRecords 方法按照路径去读取以下文 件

EF_ICCID 0x2fe2 path: 3F00
EF_MSISDN 0x6f40 path: 3F007F10
EF_MBI 0x6fc9 path: 3F007F20
EF_AD 0x6fad path: 3F007F20
EF_MWIS 0x6fca path: 3F007F20
EF_VOICE_MAIL_INDICATOR_CPHS 0x6f11 path: 3F007F20
EF_CFIS 0x6fcb path: 3F007F20
EF_CFF_CPHS 0x6f13 path: 3F007F20
EF_SPN 0x6f46 path: 3F007F20
EF_SPDI 0x6fcd path: 3F007F20
EF_PNN 0x6fc5 path: 3F007F20
EF_SST 0x6f38 path: 3F007F20
EF_INFO_CPHS 0x6f16 path: 3F007F20
EF_CSP_CPHS 0x6f15 path: 3F007F20
ECC 0x6fb7 path: 3F007FFF