# Rild.c

1. RIL\_startEventLoop

在ril.cpp中开始监听事件

1. RIL\_init

将ril.cpp中函数地址传递给qcril.c，同时获取到qcril.c中的函数地址

1. RIL\_register

将通过RIL\_init获取到的函数地址传递给ril.cpp

# QCRIL.c

qcril\_init\_hash\_table:将qcril\_event\_table指针赋值给qcril\_hash\_table，从而在qcril\_hash\_table\_lookup中将对应id的handler赋值给entry\_ptr->handler

### 主动请求流程

OnRequest → qmi\_ril\_fw\_android\_request\_render\_execution → qcril\_execute\_event → qcril\_dispatch\_event → 上面的过程通过entry\_ptr->handler调用当前Event的处理函数((qcril.c)qcril\_event\_table 中对应的函数)，进入QMI。

**主动请求结束**

执行请求函数中的回调函数 → qcril\_event\_queue → ret = write (qcril\_event.fdWakeupWrite, " ", 1) → 跳转到qcril\_event\_main n = select(qcril\_event.fdWakeupRead + 1, &rfds, NULL, NULL, NULL)

RIL\_Init → (qcril\_event.c)qcril\_event\_init → qcril\_event\_main → 等待事件 → (qcril.c)qcril\_process\_event → qcril\_execute\_event(return false,why) → qcril\_send\_request\_response → qmi\_ril\_fw\_send\_request\_response\_epilog → qcril\_response\_api[ instance\_id ] → OnRequestComplete → ril.cpp RIL\_onRequestComplete → sendResponse → sendResponseRaw 发送到RILJ的RILReceiver中

qcril\_response\_api是在RIL\_INIT中初始化的

**主动上报流程**

qmi\_client\_init\_instance函数中的回调函数 → qcril\_event\_queue → ret = write (qcril\_event.fdWakeupWrite, " ", 1) → 跳转到qcril\_event\_main n = select(qcril\_event.fdWakeupRead + 1, &rfds, NULL, NULL, NULL)

RIL\_Init → qcril\_event\_init → qcril\_event\_main → 等待事件 → qcril\_process\_event → qcril\_execute\_event → qcril\_dispatch\_event → entry\_ptr->handler → 调用qcril\_event\_table 中事件对应的处理函数 →.... → qcril\_send\_unsol\_response → qcril\_send\_unsol\_response\_epilog → qcril\_response\_api[ param\_ptr->instance\_id ] → OnUnsolicitedResponse → ril.cpp RIL\_onUnsolicitedResponse → ril\_unsol\_commands.h responseFunction /sendResponse → sendResponseRaw 发送到RILJ的RILReceiver中

qcril\_response\_api是在RIL\_INIT中初始化的

**QMI到MODEM**

client\_info.qmi\_svc\_clients[xxx]是通过qmi\_client\_init\_instance进行赋值的。

(qcril\_qmi\_client.c qcril\_qmi\_client\_send\_msg\_async → qmi\_ril\_platform\_dep.c qmi\_client\_send\_msg\_async\_with\_shm → 如果回调函数使用共公的就走这条线，如果不是，就直接调用后面的函数) → qmi\_client.c qmi\_client\_send\_msg\_async → qmi\_service.c qmi\_service\_send\_msg → qmi\_qmux\_if.c qmi\_qmux\_if\_send\_qmi\_msg → qmi\_qmux\_if\_send\_to\_qmux

1,如果定义了QMI\_MSGLIB\_MULTI\_PD

qmi\_qmux\_if\_send\_to\_qmux中出入conn\_id，并转换为qmi\_qmux.c qmi\_qmux\_tx\_msg的参数msg，在qmi\_qmux\_tx\_msg中将传入参数msg转换，作为qmi\_qmux\_tx\_to\_modem的参数，传入，在函数qmi\_qmux\_tx\_to\_modem中调用QMI\_QMUX\_IO\_PLATFORM\_SEND\_QMI\_MSG，从而进入qmi\_platform\_qmux\_io.c linux\_qmi\_qmux\_io\_send\_qmi\_msg函数中，通过传入的conn\_id参数在linux\_qmi\_qmux\_io\_conn\_info数组中找到对应的conn\_info，最终调用ret = write(conn\_info->f\_desc, (void\*) msg\_ptr, (size\_t)msg\_len)向 conn\_info->f\_desc中写数据

2, 如果没定义QMI\_MSGLIB\_MULTI\_PD

QMI\_QMUX\_IF\_PLATFORM\_TX\_MSG -> linux\_qmi\_qmux\_if\_client\_tx\_msg -> send发送数据。在linux\_qmi\_qmux\_if\_server.c文件的入口main（）函数，通过一个select来监听所有从linux\_qmi\_client端发出的socket，通过for循环调用linux\_qmi\_qmux\_if\_server\_process\_client\_msg（）处理这些监听的消息。进入到函数linux\_qmi\_qmux\_if\_server\_process\_client\_msg（）后，通过recv函数将监听的socket的消息写入buf\_size这个buffer里面，调用qmi\_qmux\_tx\_msg，在函数qmi\_qmux\_tx\_to\_modem中调用QMI\_QMUX\_IO\_PLATFORM\_SEND\_QMI\_MSG，从而进入qmi\_platform\_qmux\_io.c linux\_qmi\_qmux\_io\_send\_qmi\_msg函数中，通过传入的conn\_id参数在linux\_qmi\_qmux\_io\_conn\_info数组中找到对应的conn\_info，最终调用ret = write(conn\_info->f\_desc, (void\*) msg\_ptr, (size\_t)msg\_len)向 conn\_info->f\_desc中写数据

**QCRIL QMI Client　注册和Callback调用过程**

qcril\_qmi\_client.c qcril\_qmi\_client\_init(传入的callback函数在rx\_cb中调用) → qcril\_qmi\_init\_core\_client\_handles → qmi\_cci\_common.c qmi\_client\_init\_instance → qmi\_client\_init → qmi\_cci\_client\_alloc, clnt->info.client.ind\_cb = ind\_cb将ind\_cb赋值给clnt对象，返回qmi\_client\_init,调用xport->ops->open(...),qmi\_cci\_xport\_s → qmi\_cci\_xport\_ops\_s → 搜索qmi\_cci\_xport\_ops\_type进入qmi\_cci\_xport\_qmuxd.c　xport\_open函数

**A** qmuxd\_ref\_count\_up → qmi\_qmux\_if\_pwr\_up\_init　将rx\_cb函数传递给modem → qmi\_qmux\_if\_pwr\_up\_init\_ex → qmi\_qmux\_if\_send\_if\_msg\_to\_qmux → qmi\_qmux\_if\_send\_to\_qmux同**QCRIL到MODEM。**

**B** 将传入的参数clnt添加到client\_list，供rx\_cb调用

**qmi\_mmode\_task-modem**

1, qmi\_mmode\_task.c qmi\_mmode\_task -> qmi\_mmodei\_init -> qmi\_mmode\_svc\_init –> (qmi\_voice.c) 在qmi\_voice\_init 中注册qmi\_voice\_fw\_init\_cback 等callback, 在callback中根据QMI\_MMODE\_FW\_INIT\_CB进入qmi\_voicei\_fw\_init\_cback\_hdlr -> (Qmi\_voice\_cm\_if.c)qmi\_voice\_cm\_if\_init 通过 qmi\_mmode\_set\_cmd\_handler，将QMI\_MMODE\_CMD\_VOICE\_CM\_IF\_CB和qmi\_voice\_cm\_if\_process\_cmd添加到qmi\_mmodei\_cmd\_handler中。

2, qmi\_voice\_init中qmi\_voicei\_cmd\_callbacks的调用逻辑

qmi\_modem\_task -> qmi\_modem\_task.c qmii\_init -> ds\_qmux.c qmux\_init(方法完成对控制通道的初始化后，通过函数qmuxi\_process\_rx\_sig方法开始从共享内存接收数据) -> qmi\_set\_sig\_handler(QMI\_QMUX\_RX\_SIGNAL, qmuxi\_process\_rx\_sig,NULL)，接收到4中的QMI\_QMUX\_RX\_SIGNAL后调用

ds\_qmux.c qmuxi\_process\_rx\_sig -> qmuxi\_process\_msg -> qmuxi\_input -> qmi\_framework.c qmi\_framework\_svc\_recv -> svc\_info->cfg.cbs.cmd\_hdlr -> qmi\_voice.c qmi\_voice\_fw\_req\_cback -> QMI\_MMODE\_CMD\_VOICE\_FW\_CB -> qmi\_voice\_handle\_fw\_cmd -> QMI\_MMODE\_FW\_REQ\_CB -> qmi\_voicei\_fw\_req\_hdlr –> qmi\_mmode\_svc.c qmi\_mmode\_svc\_req\_hdlr 将cmd\_type赋值给x\_p->cmd\_list->cmd\_type –> qmi\_mmode\_svci\_dispatch\_transaction –> qmi\_mmode\_svci\_input 通过cmd\_type从qmi\_voicei\_cmd\_callbacks数组中找到对应的handler –> cmd\_hdlr->request\_hdlr

3, qmi\_voicei\_cmd\_callbacks中对应消息的处理函数-> qmi\_mmode\_send\_cmd 将QMI\_MMODE\_CMD\_VOICE\_CM\_IF\_CB封装到qmi\_mmodei\_cmd\_q中，同时发出QMI\_MMODE\_CMD\_Q\_SIGNAL信号，进入qmi\_mmode\_task.c中，调用qmi\_mmodei\_process\_cmd，从qmi\_mmodei\_cmd\_q中取出刚添加的cmd，调用qmi\_mmodei\_cmd\_handler[cmd\_ptr->cmd]( cmd\_ptr->cmd, cmd\_ptr->user\_data\_ptr )，进入A中配置的callback

4,QMI\_QMUX\_RX\_SIGNAL

qmuxi\_init -> qmuxi\_sio\_rx\_cb -> qmi\_set\_signal( QMI\_QMUX\_RX\_SIGNAL ) 进入第2步中的qmuxi\_process\_rx\_sig

5, MO call

qmi\_voice\_cm\_if.c qmi\_voice\_cm\_if\_dial\_call -> cm\_mm\_call\_cmd\_orig\_exten3/4 -> cm\_mm\_call\_cmd\_orig\_internal → cmtask.c cm\_cmd\_queue → cm\_client\_cmd\_q\_put → cm\_cmd\_q\_put → CM\_CLIENT\_CMD\_Q\_SIG → cm\_task → cmtask\_handle\_sigs → cm\_client\_cmd\_handle 调用cm\_client\_cmd\_q\_peek 获取 cmd\_ptr(根据不同的功能调用不同的proc函数) → cmcall.c cmcall\_client\_cmd\_proc → cmcall\_client\_orig\_proc → cmcall\_orig\_start → cmcall\_force\_mode\_and\_send\_event → cmcall\_event/cmcall\_orig\_proc

**A:** cmcall\_event → cmcall\_event\_2 → cmclient.c cmclient\_list\_call\_event\_ntfy → cm\_client\_call\_event\_ntfy → client\_ptr->call\_event\_func(....)其中client\_ptr是从cmclient\_map\_arr中根据clientid获取的。cmclient\_map\_arr在cmclient\_map\_init中初始化，在cm\_client\_init中对应id的进行初始化。最终进入qmi\_voice\_cm\_if\_call\_evt\_cb

callback: qmi\_voice\_cm\_if.c qmi\_voice\_cm\_if\_init -> cm\_mm\_client\_call\_reg

(CM\_CLIENT\_CMD\_CALL\_REG) →cm\_cmd\_queue -> cmtask.c cm\_client\_cmd\_handle → cmclient.c cmclient\_client\_cmd\_proc → client\_ptr->call\_event\_func = reg\_data->event\_func;

**B:**cmcall\_orig\_proc → cmxcall.c cmxcall\_send\_orig → cmcall\_send\_mc\_orig →cmxll.c cm\_mc\_send\_cmd → mc.c mc\_cmd

**Modem incoming call:**

mccccim.c mccccim\_incoming\_call → mccdma.c mcc\_cm\_mc\_rpt → cmxll.c cm\_mc\_rpt → CM\_RPT\_Q\_SIG → cmtask.c cm\_task → cmtask\_handle\_sigs → cm\_rpt\_handle → cm\_xrpt\_handle → cmcall.c cmcall\_mc\_rpt\_proc → cmxcall.c cmxcall\_mc\_rpt\_proc → **A** → Qmi\_voice\_cm\_if.c qmi\_voice\_cm\_if\_call\_evt\_cb -> QMI\_MMODE\_CMD\_VOICE\_CM\_IF\_CB -> qmi\_voice\_cm\_if\_process\_cmd -> qmi\_voice\_cm\_if\_call\_evt\_cb\_handler -> qmi\_voice\_cm\_if\_call\_event\_incom -> qmi\_voice\_call\_list.c qmi\_voice\_call\_list\_report\_all\_call\_status\_changed -> Qmi\_voice\_cm\_if.c qmi\_voice\_cm\_if\_unsol\_response -> QMI\_MMODE\_CMD\_VOICE\_CB -> qmi\_voice.c qmi\_voice\_process\_cmd -> qmi\_voicei\_all\_call\_status\_ind -> qmi\_voicei\_send\_indication -> qmi\_mmode\_svc.c qmi\_mmode\_svc\_send\_indication -> qmi\_framework.c qmi\_framework\_svc\_send\_ind -> qmi\_frameworki\_svc\_send 传入cmd.id 为QMI\_CMD\_FRAMEWORK\_SEND\_RESPONSE -> qmi\_framework\_process\_svc\_send

或者 qmi\_modem\_task\_svc.c qmi\_send\_cmd(QMI\_CMD, cmd\_ptr) -> qmi\_modem\_taski.c qmii\_process\_cmd -> qmii\_cmd\_handler[cmd\_ptr->cmd]( cmd\_ptr->cmd, cmd\_ptr->user\_data\_ptr ) cmd\_ptr->cmd为QMI\_CMD -> qmi\_cmd\_process.c qmi\_process\_cmd 在qmux\_init中通过qmi\_set\_cmd\_handler(QMI\_CMD, qmi\_process\_cmd)将QMI\_CMD和qmi\_process\_cmd绑定在一起，根据cmd.id 调用qmi\_framework.c qmi\_framework\_process\_svc\_send -> qmi\_frameworki\_qmux\_send -> ds\_qmux.c qmux\_sio\_send -> sio.c sio\_transmit/ sio\_control\_transmit消息就从modem侧发出来了

**QCRLC消息接收**

在qmi\_qmux.c的qmi\_qmux\_pwr\_up\_init方法中, qmi\_qmux\_rx\_msg会接收处理BP侧回发的消息

linux\_qmi\_qmux\_if\_server.c main() -> qmi\_qmux\_pwr\_up\_init -> QMI\_QMUX\_IO\_PLATFORM\_PWR\_UP\_INIT -> qmi\_platform\_qmux\_io.c linux\_qmi\_qmux\_io\_pwr\_up\_init进行如下的赋值：

linux\_qmi\_qmux\_io\_rx\_cb = rx\_cb\_ptr;

linux\_qmi\_qmux\_io\_event\_cb = event\_cb\_ptr;

通过1和2两个途径进入3

1,linux\_qmi\_qmux\_if\_server\_open\_port

2,linux\_qmi\_qmux\_if\_server\_process\_client\_msg -> qmi\_qmux\_tx\_msg

3,qmi\_qmux.c qmi\_qmux\_open\_connection –> QMI\_QMUX\_IO\_PLATFORM\_OPEN\_CONN -> linux\_qmi\_qmux\_io\_rx\_msg -> linux\_qmi\_qmux\_io\_rx\_cb -> linux\_qmi\_qmux\_io\_rx\_cb -> qmi\_qmux\_rx\_msg -> qmi\_qmux\_rx\_client\_msg -> qmi\_qmux\_if\_send\_to\_client -> QMI\_QMUX\_IF\_PLATFORM\_RX\_MSG -> linux\_qmi\_qmux\_if\_server\_tx\_msg -> send发送消息。

Client:

Regist 流程：

qmi.c qmi\_init -> qmi\_service.c qmi\_service\_pwr\_up\_init -> qmi\_qmux\_if.c qmi\_qmux\_if\_pwr\_up\_init -> qmi\_qmux\_if\_pwr\_up\_init\_ex -> QMI\_QMUX\_IF\_PLATFORM\_CLIENT\_INIT -> linux\_qmi\_qmux\_if\_client.c linux\_qmi\_qmux\_if\_client\_init -> 创建线程回调linux\_qmi\_qmux\_if\_rx\_msg，for语句等待数据，调用recv获取

Handle 流程：

linux\_qmi\_qmux\_if\_rx\_msg通过调用recv获取3中发送的数据，qmi\_qmux\_if\_rx\_msg