目录

[第1章 概述 2](#_Toc408587388)

[1.1 高通平台QMI通信机制 2](#_Toc408587389)

[1.2 QMI实现NV读写功能扩展产生的背景 2](#_Toc408587389)

[1.3 QMI实现NV读写功能扩展实现的要求 2](#_Toc408587390)

[第2章 流程 4](#_Toc408587391)

[2.1 QRD平台用qmi实现ap到modem之间的通信流程机制 4](#_Toc408587392)

[2.2 实现流程图…………………………………………………………………………………4](#_Toc408587392)

[第3章 实现过程 6](#_Toc408587393)

[3.1 AP端与MODEM端涉及到的修改文件 6](#_Toc408587394)

[3.2 AP端与MODEM端调试过程 6](#_Toc408587395)

[3.2.1 调试过程中遇到的问题点说明 6](#_Toc408587395)

[3.3在app层如何调用QcNvItems.java类里接口 9](#_Toc408587396)

[第4章 总结 11](#_Toc408587397)

# 概述

## 高通平台QMI通信机制

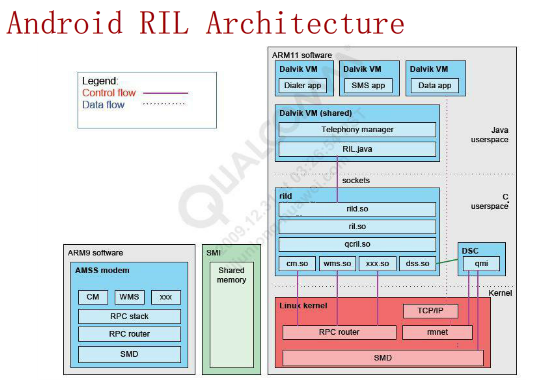
对于高通平台，RILD和Modem侧的通信是通过RPC和QMI实现的；

RPC用于多CPU之间的通信，一侧的CPU应用在ONC RPC栈中通过XDR将消息编码成字节流，通过RPC router、共享内存对应传到另一侧的CPU RPC router、ONC RPC中，XDR解码后，得到数据；

XDR是用于在同平台间实现数据连续表示的编解码机制；

Qualcomm Messaging Interface(QMI)，QMI是高通自定义的一种用于上层与底层通信的数据接口，用于终端设备的操作系统应用和底层模块的数据通信，数据业务基本用QCRIL、QMI与底层进行交互。

具体的Android RIL Architechture，即android平台机制中RIL层衔接ap与modem之间的通信流程架构图，如以下图示1所示：

**图示1**

## QMI实现NV读写功能扩展产生的背景

由于基于高通QRD8916平台的L5510项目上需要实现MES写号工具调用NV功能的开发和扩展，这个工作出发点是为了配合工具组同事那边可以快速地从modem端读取内部版本号、外部版本号、手机SN、校准标识位、功能测试标识位、Band Class等等参数信息。后续我已经将此功能在QRD8916平台上扩展为以QMI实现在ap端可以读、写modem端任意一项NV\_Item，基于高通平台，实现了我们公司自己的客制化功能：ap端到modem端NV读、写功能扩展。

## 1.3 需求功能实现的要求

QMI实现ap端到modem端通信实现要求如下：

基于QMI机制，通过我们对nv项读、写功能的扩展完善后，在ap端，如app层、frameworks层等等可以读、写modem端任意一项NV\_Item，基于高通平台，实现了我们公司自己的客制化功能：即ap端到modem端任意nv项读、写功能扩展。

ap端qcril层定义的与modem端通信下发和接收读、写消息事件id在代码中定义如下：

QCRIL\_EVT\_OEM\_BASE = 0x90000,

//20141031 zyh add for writing&reading nv data

QCRIL\_EVT\_TINNO\_RIL\_BASE = 0x91000,

QCRIL\_EVT\_TINNO\_RIL\_SYSPROP\_READ = QCRIL\_EVT\_TINNO\_RIL\_BASE + 1,

QCRIL\_EVT\_TINNO\_RIL\_SYSPROP\_WRITE = QCRIL\_EVT\_TINNO\_RIL\_BASE + 2,

QCRIL\_EVT\_TINNO\_MAX,

//20141031 zyh end

# 流程

## 2.1 QRD平台用qmi实现ap到modem之间的通信流程机制

首先，QRD8916平台用qmi实现ap到modem之间的通信流程机制如下：

统一使用QcRilHook.java类里的sendQcrilHookMsg()发消息到qcril层再到modem层，这里追述下去的机制流程是：

(ap)QcRilHook.sendQcrilHookMsg🡪QcrilMsgTunnelIfaceManager.sendOemRilRequestRaw🡪qcril.c🡪qcril\_qmi\_oem\_socket.cc🡪(modem)qmi\_voice.c🡪qmi\_voice\_cm\_if.c 🡪qmi\_tinno\_sys\_prop.c🡪对应NV值修改。

**2.2 实现流程图**

具体的功能实现流程图如图示2所示：

 **图示2**

# 实现过程

## 3.1 AP端与MODEM端涉及到的修改文件：

在此列举下所有ap端与modem端涉及到的修改文件：

ap端涉及到的修改文件如下：

~\vendor\qcom\proprietary\qcril\qcril\_qmi\qcril.c

~\vendor\qcom\proprietary\qcril\qcril\_qmi\qcrili.h

~\vendor\qcom\proprietary\qcril\qcril\_qmi\oem\_socket\qcril\_qmi\_oem\_socket.cc

~\vendor\qcom\proprietary\qmi\services\voice\_service\_v02.c

~\vendor\qcom\proprietary\qmi\services\voice\_service\_v02.h

~\vendor\qcom\proprietary\qcNvItems\src\com\qualcomm\qcnvitems\QcNvItems.java

modem端涉及到的修改文件如下：

~\modem\_proc\mmcp\mmode\qmi\src\qmi\_tinno\_sys\_prop.c

~\modem\_proc\mmcp\mmode\qmi\build\mmode\_qmi.scons

~\modem\_proc\mmcp\mmode\qmi\src\qmi\_voice\_cm\_if.c

~\modem\_proc\mmcp\mmode\qmi\src\qmi\_voice\_cm\_if.h

~\modem\_proc\mmcp\mmode\qmi\src\qmi\_voice.c

## 3.2 AP端与MODEM端调试过程：

## 3.2.1调试过程中遇到的问题点说明：

QXDM log出现QMI\_INTERNAL\_ERROR时，一定要检查下，修改了vendor\qcom\proprietary\qmi\services\层下的voice\_service\_v02.c和voice\_service\_v02.h文件后，vendor\qcom\proprietary\qmi层和vendor\qcom\proprietary\qcril层模块是否都做了mm编译，另外，这2层路径下代码模块编译后，生成的.so和库文件如下：

~/out/…/system/vendor/lib路径下生成的.so文件有：

libqcci\_legacy.so

libqmi.so

libqmi\_client\_helper.so

libqmi\_client\_qmux.so

libqmi\_csvt\_srvc.so

libqmiservices.so

libril-qc-qmi-1.so

~/out/…/system/bin路径下生成的文件有：

qmi\_simple\_ril\_test

qmiproxy

qmuxd

在调试的时候，以上列举的这10个文件一定都要push到手机中，有可能.h文件的修改没有编译到，以免ap端到modem端通信失败，导致nv修改失败；通常使用make systemimage编译的话，测试就没有问题，如果条件允许的话，最好make全工程；

voice\_service\_v02.c里面定义的4个结构体：

static const uint8\_t voice\_set\_config\_req\_msg\_data\_v02[]

static const uint8\_t voice\_set\_config\_resp\_msg\_data\_v02[]

static const uint8\_t voice\_get\_config\_req\_msg\_data\_v02[]

static const uint8\_t voice\_get\_config\_resp\_msg\_data\_v02[]

和voice\_service\_v02.h里定义的4个结构体：

voice\_set\_config\_req\_msg\_v02

voice\_set\_config\_resp\_msg\_v02

voice\_get\_config\_req\_msg\_v02

voice\_get\_config\_resp\_msg\_v02

它们的成员数目和顺序一定要相同；

也就是说，在voice\_service\_v02.c里定义的4个结构体里添加成员时，

static const uint8\_t voice\_set\_config\_req\_msg\_data\_v02[]

static const uint8\_t voice\_set\_config\_resp\_msg\_data\_v02[]

static const uint8\_t voice\_get\_config\_req\_msg\_data\_v02[]

static const uint8\_t voice\_get\_config\_resp\_msg\_data\_v02[]

要参照voice\_service\_v02.h里下面4个结构体

voice\_set\_config\_req\_msg\_v02

voice\_set\_config\_resp\_msg\_v02

voice\_get\_config\_req\_msg\_v02

voice\_get\_config\_resp\_msg\_v02

的成员顺序，如果我们新加的成员是位于结构体最后，voice\_service\_v02.c里的原有的源代码里标识最后一个成员的QMI\_IDL\_TLV\_FLAGS\_LAST\_TLV标识位一定要去掉，此标识位QMI\_IDL\_TLV\_FLAGS\_LAST\_TLV相应的加到我们新添加的成员里，代码如下：

添加成员前,最后一个成员如下：

QMI\_IDL\_TLV\_FLAGS\_LAST\_TLV | QMI\_IDL\_TLV\_FLAGS\_OPTIONAL | (QMI\_IDL\_OFFSET8(voice\_set\_config\_req\_msg\_v02, voice\_domain) - QMI\_IDL\_OFFSET8(voice\_set\_config\_req\_msg\_v02, voice\_domain\_valid)),

0x15,

QMI\_IDL\_1\_BYTE\_ENUM,

QMI\_IDL\_OFFSET8(voice\_set\_config\_req\_msg\_v02, voice\_domain)

添加我们新加的成员后，代码修改如下：

QMI\_IDL\_TLV\_FLAGS\_OPTIONAL | (QMI\_IDL\_OFFSET8(voice\_set\_config\_req\_msg\_v02, voice\_domain) - QMI\_IDL\_OFFSET8(voice\_set\_config\_req\_msg\_v02, voice\_domain\_valid)),

0x15,

QMI\_IDL\_1\_BYTE\_ENUM,

QMI\_IDL\_OFFSET8(voice\_set\_config\_req\_msg\_v02, voice\_domain)

//20141031 zyh add for writing nv data to modem

,QMI\_IDL\_TLV\_FLAGS\_LAST\_TLV | QMI\_IDL\_TLV\_FLAGS\_OPTIONAL | (QMI\_IDL\_OFFSET8(voice\_set\_config\_req\_msg\_v02, sys\_prop\_data) - QMI\_IDL\_OFFSET8(voice\_set\_config\_req\_msg\_v02, sys\_prop\_data\_valid)),

0x4F,

QMI\_IDL\_FLAGS\_IS\_ARRAY| QMI\_IDL\_GENERIC\_1\_BYTE | QMI\_IDL\_FLAGS\_SZ\_IS\_16,

QMI\_IDL\_OFFSET8(voice\_set\_config\_req\_msg\_v02, sys\_prop\_data),

00,02

//20141031 zyh end

voice\_service\_v02.c文件有2处代码一定记得修改：

voice\_service\_command\_messages\_v02[] = {

……………

//zyh change 39 to 1024

{QMI\_VOICE\_SET\_CONFIG\_REQ\_V02, QMI\_IDL\_TYPE16(0, 53), 1024},

…………..

}

voice\_service\_response\_messages\_v02[] = {

//zyh changed 55 to 1024 as voice\_get\_config\_resp\_msg\_v02 is added more fields.

{QMI\_VOICE\_GET\_CONFIG\_RESP\_V02, QMI\_IDL\_TYPE16(0, 56), 1024},

…………

}

## 3.3在ap层如何调用QcNvItems.java类里接口

ap层最终都是通过qcril层发消息到modem，通过起QcrilMsgTunnelService来发消息, 如果是在开机运行到的地方调用QcNvItems.java类接口，需要设个调用延迟处理，避免开机运行地方过早，QcrilMsgTunnelService在调用的地方还没有加载起来，实例测试代码如下：

添加处~\packages\apps\Mms\src\com\android\mms\MmsApp.java中onCreat(){}，

……

//20141113 zyh add,test QcRilHook R&W NV

private void set\_factory\_info()

{

try {

String factory\_info = mQcNvItems.getSysPropInfo(114);

Log.d(LOG\_TAG, "zyh factory\_info = " + factory\_info);

mQcNvItems.setSysPropInfo(114, "abcdefgh", "000000");

}catch (Exception e) {

Log.e(LOG\_TAG, "zyh catch error Exception while" + e);

}

}

……

public void onCreate() {

……

mQcNvItems = new QcNvItems(context);

new Thread(){

public void run(){

try {

Thread.sleep(1000);

set\_factory\_info();

} catch (Exception e) {

Log.e(LOG\_TAG, "zyh catch error Exception while sleep delay failed." + e);

}

}

}.start();

//20141113 zyh end

……

}

# 总结

由于高通平台只是在qcril层实现了ap端到modem的QMI通信机制，各家OEM或ODM就需要在此机制基础上，自行实现自己的通信扩展机制及客制化需求。如此一来，就需要各家OEM或ODM自己定义上层与modem端通信相匹配的id(本次扩展的读、写消息事件id，分别是0x91001和0x91002)以及下发、接收消息载体的结构体，用来向modem层发消息及接收处理modem层上报上来的消息和消息内容体。

目前存在的不足是：

1、对于modem端的任意NV项的内容的读、写，目前ap层定义的结构体里用来接收通信内容消息的只是char型的，所以对于modem端的不是char型内容的NV项，需要在modem端进行char型的转换后再封装传给上层ap；

2、由于高通各个平台以及同一平台各个release基线版本之间都会存在差异，所以针对其他高通平台的话，我们在功能实现代码移植的时候需要找出平台间QMI机制的差异化。