Location分析：

LocApiV02.cpp (vendor1\vendor\qcom\opensource\location\loc\_api\loc\_api\_v02) 120191 2016-06-29

LocationManager.java (frameworks\base\location\java\android\location) 69697 2016-06-29

//发送message相关

private static final int TYPE\_LOCATION\_CHANGED = 1;

private static final int TYPE\_STATUS\_CHANGED = 2;

private static final int TYPE\_PROVIDER\_ENABLED = 3;

private static final int TYPE\_PROVIDER\_DISABLED = 4;

@Override

public void onStatusChanged(String provider, int status, Bundle extras) {

Message msg = Message.obtain();

msg.what = TYPE\_STATUS\_CHANGED;

b.putString("provider", provider);

b.putInt("status", status);

msg.obj = b;

mListenerHandler.sendMessage(msg);

}

private void \_handleMessage(Message msg) {

switch (msg.what) {

case TYPE\_LOCATION\_CHANGED:

Location location = new Location((Location) msg.obj);

mListener.onLocationChanged(location);

break;

}

收据接受相关：

Nmea(long timestamp, String nmea) {

mTimestamp = timestamp;

mNmea = nmea;

}

public void onNmeaReceived(long timestamp, String nmea) {

if (mNmeaListener != null) {

synchronized (mNmeaBuffer) {

mNmeaBuffer.add(new Nmea(timestamp, nmea));

}

Message msg = Message.obtain();

msg.what = NMEA\_RECEIVED;

// remove any NMEA\_RECEIVED messages already in the queue

mGpsHandler.removeMessages(NMEA\_RECEIVED);

mGpsHandler.sendMessage(msg);

}

}

private final Handler mGpsHandler = new Handler() {

@Override

public void handleMessage(Message msg) {

if (msg.what == NMEA\_RECEIVED) {

synchronized (mNmeaBuffer) {

int length = mNmeaBuffer.size();

for (int i = 0; i < length; i++) {

Nmea nmea = mNmeaBuffer.get(i);

mNmeaListener.onNmeaReceived(nmea.mTimestamp, nmea.mNmea);

}

mNmeaBuffer.clear();

}

} else {

// synchronize on mGpsStatus to ensure the data is copied atomically.

synchronized(mGpsStatus) {

mListener.onGpsStatusChanged(msg.what);

}

public boolean addNmeaListener(GpsStatus.NmeaListener listener) {

boolean result;

result = mService.addGpsStatusListener(transport, mContext.getPackageName());

if (result) {

mNmeaListeners.put(listener, transport);

}

}

Loc\_eng.cpp (hardware\qcom\gps\loc\_api\libloc\_api\_50001) 101465 2016-06-29

void LocEngReportSv::proc() const {

//loc eng层是通过一个proc()的方法把数据传递到上层的

loc eng层是通过一个proc()的方法把数据传递到上层的

LocEngAdapter\* adapter = (LocEngAdapter\*)mAdapter;

loc\_eng\_data\_s\_type\* locEng = (loc\_eng\_data\_s\_type\*)adapter->getOwner();

if (locEng->mute\_session\_state != LOC\_MUTE\_SESS\_IN\_SESSION)

{

if (locEng->sv\_status\_cb != NULL) {

locEng->sv\_status\_cb((GpsSvStatus\*)&(mSvStatus),

(void\*)mSvExt);

}

if (locEng->generateNmea)

{

loc\_eng\_nmea\_generate\_sv(locEng, mSvStatus, mLocationExtended);

}

}

}

LocApiV02.cpp (vendor1\vendor\qcom\opensource\location\loc\_api\loc\_api\_v02) 120191 2016-06-29

/\* Initialize a loc api v02 client AND

check which loc message are supported by modem \*/

enum loc\_api\_adapter\_err

LocApiV02 :: open(LOC\_API\_ADAPTER\_EVENT\_MASK\_T mask)

{

/\* initialize the loc api v02 interface, note that

the locClientOpen() function will block if the

service is unavailable for a fixed time out \*/

// it is important to cap the mask here, because not all LocApi's

// can enable the same bits, e.g. foreground and bckground.

status = locClientOpen(adjustMaskForNoSession(qmiMask), &globalCallbacks,

&clientHandle, (void \*)this);

}

LocApiV02.cpp (vendor1\vendor\qcom\opensource\location\loc\_api\loc\_api\_v02) 120191 2016-06-29

locClientStatusEnumType locClientOpen (

locClientEventMaskType eventRegMask,

const locClientCallbacksType\* pLocClientCallbacks,

locClientHandleType\* pLocClientHandle,

const void\* pClientCookie)

{

while ((status = locClientOpenInstance(eventRegMask, instanceId, pLocClientCallbacks,

pLocClientHandle, pClientCookie)) != eLOC\_CLIENT\_SUCCESS) {

}

}

locClientStatusEnumType locClientOpenInstance (

locClientEventMaskType eventRegMask,

int instanceId,

const locClientCallbacksType\* pLocClientCallbacks,

locClientHandleType\* pLocClientHandle,

const void\* pClientCookie)

{

EXIT\_LOG\_CALLFLOW(%s, "loc client open");

status = locClientQmiCtrlPointInit(pCallbackData, instanceId);

//fill in the event callback

pCallbackData->eventCallback = pLocClientCallbacks->eventIndCb;

//fill in the response callback

pCallbackData->respCallback = pLocClientCallbacks->respIndCb;

}

LocApiV02.cpp (vendor1\vendor\qcom\opensource\location\loc\_api\loc\_api\_v02) 120191 2016-06-29

LocApiV02 :: open(LOC\_API\_ADAPTER\_EVENT\_MASK\_T mask)

{

status = locClientOpen(adjustMaskForNoSession(qmiMask), &globalCallbacks,

&clientHandle, (void \*)this);

}

LocApiV02.cpp (vendor1\vendor\qcom\opensource\location\loc\_api\loc\_api\_v02) 120191 2016-06-29

locClientCallbacksType globalCallbacks =

{

sizeof(locClientCallbacksType),

globalEventCb,

globalRespCb,

globalErrorCb

};

LocApiV02.cpp (vendor1\vendor\qcom\opensource\location\loc\_api\loc\_api\_v02) 120191 2016-06-29

/\* static event callbacks that call the LocApiV02 callbacks\*/

/\* global event callback, call the eventCb function in loc api adapter v02

instance \*/

static void globalEventCb(locClientHandleType clientHandle,

uint32\_t eventId,

const locClientEventIndUnionType eventPayload,

void\* pClientCookie)

{

LocApiV02 \*locApiV02Instance =

(LocApiV02 \*)pClientCookie;

locApiV02Instance->eventCb(clientHandle, eventId, eventPayload);

}

LocApiV02.cpp (vendor1\vendor\qcom\opensource\location\loc\_api\loc\_api\_v02) 120191 2016-06-29

/\* event callback registered with the loc\_api v02 interface \*/

void LocApiV02 :: eventCb(locClientHandleType clientHandle,

uint32\_t eventId, locClientEventIndUnionType eventPayload)

{

switch(eventId)

{

//Position Report

case QMI\_LOC\_EVENT\_POSITION\_REPORT\_IND\_V02:

reportPosition(eventPayload.pPositionReportEvent);

break;

// NMEA

case QMI\_LOC\_EVENT\_NMEA\_IND\_V02:

reportNmea(eventPayload.pNmeaReportEvent);

break;

// Satellite report

case QMI\_LOC\_EVENT\_GNSS\_SV\_INFO\_IND\_V02:

reportSv(eventPayload.pGnssSvInfoReportEvent);

break;

// Status report

case QMI\_LOC\_EVENT\_ENGINE\_STATE\_IND\_V02:

reportEngineState(eventPayload.pEngineState);

break;

// AGPS connection request

case QMI\_LOC\_EVENT\_LOCATION\_SERVER\_CONNECTION\_REQ\_IND\_V02:

reportAtlRequest(eventPayload.pLocationServerConnReqEvent);

break;

}

LocApiV02.cpp (vendor1\vendor\qcom\opensource\location\loc\_api\loc\_api\_v02) 120191 2016-06-29

void LocApiV02 :: reportAtlRequest(

const qmiLocEventLocationServerConnectionReqIndMsgT\_v02 \* server\_request\_ptr)

{

if(server\_request\_ptr->requestType == eQMI\_LOC\_SERVER\_REQUEST\_OPEN\_V02 )

{

AGpsType agpsType;

switch(server\_request\_ptr->wwanType)

{

case eQMI\_LOC\_WWAN\_TYPE\_INTERNET\_V02:

agpsType = AGPS\_TYPE\_WWAN\_ANY;

requestATL(connHandle, agpsType);

break;

case eQMI\_LOC\_WWAN\_TYPE\_AGNSS\_EMERGENCY\_V02:

requestSuplES(connHandle);

break;

}

}

Ulp\_msg.cpp (vendor1\vendor\qcom\proprietary\gps\ulp2\src) 45937 2016-07-01

int ulp\_msg\_process\_gnss\_position\_report (const enum loc\_sess\_status status,){

ret\_val = ulp\_brain\_process\_gnss\_position\_report (status, tech\_mask, locationPtr, locationExtendedPtr, locationExt);

if (ulp\_data.run\_provider\_selection\_logic == true)

{

ret\_val = ulp\_brain\_select\_providers ();

}

} while (0);

}

消息的发送和接受：

接受：

Ulp\_msg.cpp (vendor1\vendor\qcom\proprietary\gps\ulp2\src) 45937 2016-07-01

void ulp\_msg\_main(void \* context)

{

while (1)

{

msq\_q\_err\_type result = msg\_q\_rcv(ulp\_data.loc\_proxy->mQ,

(void \*\*) &msg);

}

}

发送

int ulp\_msg\_send\_monitor\_request ()

{

msg\_q\_snd(ulp\_data.loc\_proxy->mQ,

ulpMonitorMsg,

ulp\_msg\_free);

接收 发送：

MsgTask.cpp (hardware\qcom\gps\core) 4625 2016-06-29

void\* MsgTask::loopMain(void\* arg) {

MsgTask\* copy = (MsgTask\*)arg;

while (1) {//发送

msq\_q\_err\_type result = msg\_q\_rcv((void\*)copy->mQ, (void \*\*)&msg);

}

msg->log();

// there is where each individual msg handling is invoked

msg->proc();

}

发送：

void MsgTask::sendMsg(const LocMsg\* msg) const {

msg\_q\_snd((void\*)mQ, (void\*)msg, LocMsgDestroy);

}

MsgTask.cpp (hardware\qcom\gps\core) 4625 2016-06-29

MsgTask::MsgTask(tAssociate tAssociator, const char\* threadName) :

mQ(msg\_q\_init2()), mAssociator(tAssociator){

createPThread(threadName);

}

void MsgTask::createPThread(const char\* threadName) {

if (!pthread\_create(&tid, &attr, loopMain,

(void\*)new MsgTask(mQ, mAssociator)) &&

NULL != threadName)

}

MsgTask的调用和创建：

Loc\_eng.cpp (hardware\qcom\gps\loc\_api\libloc\_api\_50001) 101465 2016-06-29

int loc\_eng\_init(loc\_eng\_data\_s\_type &loc\_eng\_data, LocCallbacks\* callbacks,

LOC\_API\_ADAPTER\_EVENT\_MASK\_T event, ContextBase\* context)

{

loc\_eng\_data.adapter =

new LocEngAdapter(event, &loc\_eng\_data, context,

(MsgTask::tCreate)callbacks->create\_thread\_cb);

}

LocApiV02.cpp (vendor1\vendor\qcom\opensource\location\loc\_api\loc\_api\_v02) 120191 2016-06-29

LocApiBase\* getLocApi(const MsgTask \*msgTask,

LOC\_API\_ADAPTER\_EVENT\_MASK\_T exMask,

ContextBase\* context)

{

return new LocApiV02(msgTask, exMask, context);

}

LocApiV02 :: LocApiV02(const MsgTask\* msgTask,

LOC\_API\_ADAPTER\_EVENT\_MASK\_T exMask,

ContextBase\* context):

LocApiBase(msgTask, exMask, context),

clientHandle(LOC\_CLIENT\_INVALID\_HANDLE\_VALUE),

dsClientHandle(NULL), mGnssMeasurementSupported(sup\_unknown),

mQmiMask(0), mInSession(false), mEngineOn(false)

{

// initialize loc\_sync\_req interface

loc\_sync\_req\_init();

}

LocApiBase.cpp (hardware\qcom\gps\core) 18034 2016-06-29

LocApiBase::LocApiBase(const MsgTask\* msgTask,

LOC\_API\_ADAPTER\_EVENT\_MASK\_T excludedMask,

ContextBase\* context) :

mExcludedMask(excludedMask), mMsgTask(msgTask),

mMask(0), mSupportedMsg(0), mContext(context)

{

}

LocEngAdapter.cpp (hardware\qcom\gps\loc\_api\libloc\_api\_50001) 13192 2016-06-29

LocEngAdapter::LocEngAdapter(LOC\_API\_ADAPTER\_EVENT\_MASK\_T mask,

void\* owner, ContextBase\* context,

MsgTask::tCreate tCreator) :

LocAdapterBase(mask,

context == NULL?

LocDualContext::getLocFgContext(tCreator,

LocDualContext::mLocationHalName)

:context),

LocDualContext.cpp (hardware\qcom\gps\core) 7439 2016-06-29

ContextBase\* LocDualContext::getLocFgContext(MsgTask::tCreate tCreator,

const char\* name)

{

const MsgTask\* msgTask = getMsgTask(tCreator, name);

}

const MsgTask\* LocDualContext::getMsgTask(MsgTask::tCreate tCreator,

const char\* name)

{

mMsgTask = new MsgTask(tCreator, name);

}

MsgTask.cpp (hardware\qcom\gps\core) 4625 2016-06-29

MsgTask::MsgTask(tCreate tCreator, const char\* threadName) :

mQ(msg\_q\_init2()), mAssociator(NULL){

if (tCreator) {

tCreator(threadName, loopMain,

(void\*)new MsgTask(mQ, mAssociator));

} else {

createPThread(threadName);

}

}

Loc\_eng.cpp (hardware\qcom\gps\loc\_api\libloc\_api\_50001) 101465 2016-06-29

int loc\_eng\_init(loc\_eng\_data\_s\_type &loc\_eng\_data, LocCallbacks\* callbacks,

LOC\_API\_ADAPTER\_EVENT\_MASK\_T event, ContextBase\* context)

{

loc\_eng\_data.adapter =

new LocEngAdapter(event, &loc\_eng\_data, context,

(MsgTask::tCreate)callbacks->create\_thread\_cb);

}

RPC通道：

LocApiRpc.cpp (hardware\qcom\gps\loc\_api\libloc\_api-rpc-50001\libloc\_api-rpc-glue\src) 54937 2016-06-29

LocApiRpc::LocApiRpc(const MsgTask\* msgTask,

LOC\_API\_ADAPTER\_EVENT\_MASK\_T exMask,

ContextBase\* context) :

LocApiBase(msgTask, exMask, context),

client\_handle(RPC\_LOC\_CLIENT\_HANDLE\_INVALID),

dataEnableLastSet(-1)

{

memset(apnLastSet, 0, sizeof(apnLastSet));

loc\_api\_glue\_init();//初始化，创建 create RPC

}

getLocApi函数的调用：

ContextBase.cpp (hardware\qcom\gps\core) 4370 2016-06-29

LocApiBase\* ContextBase::createLocApi(LOC\_API\_ADAPTER\_EVENT\_MASK\_T exMask)

{

if((handle = dlopen("libloc\_api\_v02.so", RTLD\_NOW)) != NULL) {

getLocApi\_t\* getter = (getLocApi\_t\*)dlsym(handle, "getLocApi");

if(getter != NULL) {

LOC\_LOGD("%s:%d]: getter is not NULL for LocApiV02", \_\_func\_\_, \_\_LINE\_\_);

////try to see if LocApiV02 is present

locApi = (\*getter)(mMsgTask, exMask, this);

}

}

else {

LOC\_LOGD("%s:%d]: libloc\_api\_v02.so is NOT present. Trying RPC",

\_\_func\_\_, \_\_LINE\_\_);

handle = dlopen("libloc\_api-rpc-qc.so", RTLD\_NOW);

if (NULL != handle) {

getLocApi\_t\* getter = (getLocApi\_t\*)dlsym(handle, "getLocApi");

if (NULL != getter) {

LOC\_LOGD("%s:%d]: getter is not NULL in RPC", \_\_func\_\_, \_\_LINE\_\_);

locApi = (\*getter)(mMsgTask, exMask, this);

}

}

}

// locApi could still be NULL at this time

// we would then create a dummy one

if (NULL == locApi) {

locApi = new LocApiBase(mMsgTask, exMask, this);

}

}

LocAdapterBase.cpp (hardware\qcom\gps\core) 4502 2016-06-29

LocAdapterBase::LocAdapterBase(const LOC\_API\_ADAPTER\_EVENT\_MASK\_T mask,

ContextBase\* context, LocAdapterProxyBase \*adapterProxyBase) :

mEvtMask(mask), mContext(context),

mLocApi(context->getLocApi()), mLocAdapterProxyBase(adapterProxyBase),

mMsgTask(context->getMsgTask())

{

mLocApi->addAdapter(this);

}

Ulp\_gnss.cpp (vendor1\vendor\qcom\proprietary\gps\ulp2\src) 10006 2016-07-01

int ulp\_gnss\_start\_engine ()

{

}

Ulp\_brain.cpp (vendor1\vendor\qcom\proprietary\gps\ulp2\src) 84177 2016-07-01

static int ulp\_brain\_turn\_onoff\_gnss\_provider ()

{

ulp\_gnss\_start\_engine ();

}

int ulp\_brain\_select\_providers ()

{

ulp\_brain\_turn\_onoff\_quipc\_provider ();

ulp\_brain\_turn\_onoff\_gnss\_provider ();

ulp\_brain\_turn\_onoff\_gnp\_provider ();

ulp\_brain\_turn\_onoff\_zpp\_provider ();

}

Ulp\_msg.cpp (vendor1\vendor\qcom\proprietary\gps\ulp2\src) 45937 2016-07-01

int ulp\_msg\_process\_gnss\_position\_report (const enum loc\_sess\_status status,）{

ret\_val = ulp\_brain\_process\_gnss\_position\_report (status, tech\_mask, locationPtr, locationExtendedPtr, locationExt);

if (ulp\_data.run\_provider\_selection\_logic == true)

{

ret\_val = ulp\_brain\_select\_providers ();

}

} while (0);

}

void ulp\_msg\_main(void \* context)

{

while (1)

{

//接收消息

msq\_q\_err\_type result = msg\_q\_rcv(ulp\_data.loc\_proxy->mQ,

(void \*\*) &msg);

if ( positionMsg->location.position\_source == ULP\_LOCATION\_IS\_FROM\_GNSS)

{

ulp\_msg\_process\_gnss\_position\_report ( positionMsg->status,

positionMsg->technology\_mask,

&positionMsg->location,

&positionMsg->locationExtended,

positionMsg->locationExt);

}

}

int ulp\_msg\_init ()

{

thread\_id = ulp\_data.ulp\_create\_thread\_cb("ulp\_msg",

ulp\_msg\_main,

(void \*)&ulp\_data);

}

Ulp\_main.cpp (vendor1\vendor\qcom\proprietary\gps\ulp2\src) 10917 2016-07-01

int ulp\_init(UlpEngineCallbacks\* pEngineCallbacks,

UlpNetworkLocationCallbacks\* pNetworkLocationCallbacks,

UlpPhoneContextCallbacks\* pPhoneContextCallbacks)

{

retVal = ulp\_data\_init ();

// Initialize ULP message module

retVal = ulp\_msg\_init ();

// Initialize ULP monitor module

retVal = ulp\_monitor\_init ();

// Initialize ULP QUIPC module

retVal = ulp\_quipc\_init ();

LBSProxy::ulpRequestLoc(ulp\_data.loc\_proxy);

}

Ulp\_main.cpp (vendor1\vendor\qcom\proprietary\gps\ulp2\src) 10917 2016-07-01

// Defines the ulpInterface in ulp.h

static const ulpInterface ulpInf =

{

sizeof(ulpInterface),

ulp\_init

};

extern "C" const ulpInterface\* ulp\_get\_interface ()

{

return &ulpInf;

}

Agps

Loc\_eng\_agps.cpp (hardware\qcom\gps\loc\_api\libloc\_api\_50001) 33009 2016-06-29

void DSStateMachine :: onRsrcEvent(AgpsRsrcStatus event)

{

switch (event)

{

case RSRC\_GRANTED:

LOC\_LOGD("DSStateMachine :: onRsrcEvent RSRC\_GRANTED\n");

mStatePtr = mStatePtr->onRsrcEvent(event, NULL);

break;

case RSRC\_RELEASED:

LOC\_LOGD("DSStateMachine :: onRsrcEvent RSRC\_RELEASED\n");

mStatePtr = mStatePtr->onRsrcEvent(event, NULL);

}

case RSRC\_DENIED:

mStatePtr = mStatePtr->onRsrcEvent(event, NULL);

}

Loc\_eng.cpp (hardware\qcom\gps\loc\_api\libloc\_api\_50001) 101465 2016-06-29

void LocEngSuplEsOpened::proc() const {

loc\_eng\_data\_s\_type\* locEng = (loc\_eng\_data\_s\_type\*)mLocEng;

if (locEng->ds\_nif) {

AgpsStateMachine\* sm = locEng->ds\_nif;

sm->onRsrcEvent(RSRC\_GRANTED);

}

}

void LocEngSuplEsClosed::proc() const {

loc\_eng\_data\_s\_type\* locEng = (loc\_eng\_data\_s\_type\*)mLocEng;

if (locEng->ds\_nif) {

AgpsStateMachine\* sm = locEng->ds\_nif;

sm->onRsrcEvent(RSRC\_RELEASED);

}

}

回调的使用：

com\_android\_server\_location\_GpsLocationProvider.cpp (frameworks\base\services\core\jni) 58207 2016-06-29

static void agps\_status\_callback(AGpsStatus\* agps\_status)

{

env->CallVoidMethod(mCallbacksObj, method\_reportAGpsStatus, agps\_status->type,

agps\_status->status, byteArray);

}

AGpsCallbacks sAGpsCallbacks = {

agps\_status\_callback,

create\_thread\_callback,

};

static jboolean android\_location\_GpsLocationProvider\_init(JNIEnv\* env, jobject obj)

{

if (sAGpsInterface)

sAGpsInterface->init(&sAGpsCallbacks);

}

用到回调的地方：

com\_android\_server\_location\_GpsLocationProvider.cpp (frameworks\base\services\core\jni)

58207 2016-06-29

static void sv\_status\_callback(GpsSvStatus\* sv\_status)

{

memcpy(&sGpsSvStatus, sv\_status, sizeof(sGpsSvStatus));

}

GpsCallbacks sGpsCallbacks = {

sizeof(GpsCallbacks),

location\_callback,

status\_callback,

sv\_status\_callback,

}

static jboolean android\_location\_GpsLocationProvider\_init(JNIEnv\* env, jobject obj)

{

if (!sGpsInterface || sGpsInterface->init(&sGpsCallbacks) != 0)

return JNI\_FALSE;

}

static const GpsInterface sLocEngInterface =

{

sizeof(GpsInterface),

loc\_init,

loc\_start,

}

static int loc\_init(GpsCallbacks\* callbacks)

{

gps\_sv\_cb = callbacks->sv\_status\_cb;

}

Loc.cpp (hardware\qcom\gps\loc\_api\libloc\_api\_50001) 33961 2016-06-29

static void local\_sv\_cb(GpsSvStatus\* sv\_status, void\* svExt)

{

gps\_sv\_cb(sv\_status);//最终数据来源

}

gps\_sv\_cb 分析：

static int loc\_init(GpsCallbacks\* callbacks)

{

LocCallbacks clientCallbacks = {local\_loc\_cb, /\* location\_cb \*/

callbacks->status\_cb, /\* status\_cb \*/

local\_sv\_cb, /\* sv\_status\_cb \*/

callbacks->nmea\_cb, /\* nmea\_cb \*/

callbacks->set\_capabilities\_cb, /\* set\_capabilities\_cb \*/

retVal = loc\_eng\_init(loc\_afw\_data, &clientCallbacks, event, NULL);

}

Loc\_eng.cpp (hardware\qcom\gps\loc\_api\libloc\_api\_50001) 101465 2016-06-29

int loc\_eng\_init(loc\_eng\_data\_s\_type &loc\_eng\_data, LocCallbacks\* callbacks,

LOC\_API\_ADAPTER\_EVENT\_MASK\_T event, ContextBase\* context)

{

loc\_eng\_data.location\_cb = callbacks->location\_cb;

loc\_eng\_data.sv\_status\_cb = callbacks->sv\_status\_cb;

loc\_eng\_data.status\_cb = callbacks->status\_cb;

loc\_eng\_data.adapter =

new LocEngAdapter(event, &loc\_eng\_data, context,

(MsgTask::tCreate)callbacks->create\_thread\_cb);

}

几种对应的数据：

LocApiV02.cpp (vendor1\vendor\qcom\opensource\location\loc\_api\loc\_api\_v02) 120191 2016-06-29

void LocApiV02 :: reportPosition (

const qmiLocEventPositionReportIndMsgT\_v02 \*location\_report\_ptr)

{

LOC\_LOGD("Reporting postion from V2 Adapter\n");

}

Loc\_eng.cpp (hardware\qcom\gps\loc\_api\libloc\_api\_50001) 101465 2016-06-29

LocEngReportPosition::LocEngReportPosition(LocAdapterBase\* adapter,

UlpLocation &loc,）{

}

LocUlpProxy.cpp (vendor1\vendor\qcom\proprietary\gps\ulp2\src) 6645 2016-07-01

bool LocUlpProxy::reportPosition(UlpLocation &location,

GpsLocationExtended &locationExtend{

{

ulp\_msg\_report\_position \*msg(

new ulp\_msg\_report\_position(&ulp\_data,

location,locationExtended,

locationExt,

status,

loc\_technology\_mask));

msg\_q\_snd(mQ, msg, ulp\_msg\_free);

}

基本流程分析：

MsgTask.cpp (hardware\qcom\gps\core) 4625 2016-06-29

void\* MsgTask::loopMain(void\* arg) {

MsgTask\* copy = (MsgTask\*)arg;

while (1) {

//接受消息

msq\_q\_err\_type result = msg\_q\_rcv((void\*)copy->mQ, (void \*\*)&msg);

msg->log();

// there is where each individual msg handling is invoked

msg->proc();

}

}

-->loc\_eng\_set\_position\_mode

-->loc\_eng\_start

--->ulp\_msg\_process\_gnss\_set\_pos\_mode

--->ulp\_msg\_process\_criteria\_update

--->ulp\_data\_add\_criteria

--->ulp\_brain\_process\_criteria\_update

--->ulp\_brain\_select\_providers

---->ulp\_brain\_turn\_onoff\_quipc\_provider

---->ulp\_quipc\_stop\_engine

--->ulp\_quipc\_engine\_running

--->ulp\_brain\_turn\_onoff\_gnss\_provider

--->ulp\_gnss\_start\_engine

--->ulp\_quipc\_engine\_running

--->ulp\_brain\_transition\_all\_providers

--->ulp\_quipc\_engine\_running

--->ulp\_brain\_transition\_gnss\_running

--->ulp\_brain\_transition\_all\_providers

--->ulp\_brain\_turn\_onoff\_gnp\_provider

--->ulp\_gnp\_engine\_running

--->ulp\_gnp\_stop\_engine

---->ulp\_gnp\_engine\_running

--->ulp\_brain\_turn\_onoff\_zpp\_provider

--->ulp\_zpp\_stop\_engine

--->ulp\_zpp\_engine\_running

---->ulp\_msg\_process\_start\_req

--->locClientSendReq ===QMI\_LOC\_REG\_EVENTS\_REQ\_V02

--->locClientSendReq==QMI\_LOC\_SET\_OPERATION\_MODE\_REQ\_V02

-->globalRespCb===QMI\_LOC\_SET\_OPERATION\_MODE\_REQ\_V02

--->locClientSendReq====QMI\_LOC\_START\_REQ\_V02

--->status\_cb====GPS\_STATUS\_SESSION\_BEGIN

--->GPS\_STATUS\_ENGINE\_ON===GPS\_STATUS\_SESSION\_BEGIN

Modem端分析：

loc\_middleware\_task

--->loc\_middlware\_module\_init();//中间件初始化

--->loc\_middleware\_start\_up();//建立ipc和定时器

---->os\_DogHeartbeatRegister();//看门狗

--->os\_TaskReadyAck();

--->os\_TaskStopRegister();

--->mqcsi\_powerup\_init();//初始化mqcsi模块

--->locQmiShimInit();//初始化qmi数据结构

--->slim\_TaskStart();

--->p\_msg = os\_IpcReceive();//循环解决msg

loc\_middleware\_proc

--->

loc\_middleware\_proc (p\_msg);

Gps rf的初始化

rfgnss\_mc\_configure

--->rfgnss\_mc\_configure

-->rf\_hal\_bus\_enable

--->rfgnss\_core\_rf\_card\_init (device, rfgnss\_mode);

--->rfgnss\_core\_adc\_init(device, rfgnss\_mode);

--->rfgnss\_core\_bbc\_init(rfgnss\_mode);

--->gnss\_ApqPgaCalNvReadWrite(&rfgnss\_pga\_cal\_nv, TRUE)//读取nv

--->rfgnss\_core\_program\_notch\_filter(rfgnss\_spur\_system\_det);

正常流程

//发起定位

loc\_pd\_process\_fix\_request

--->