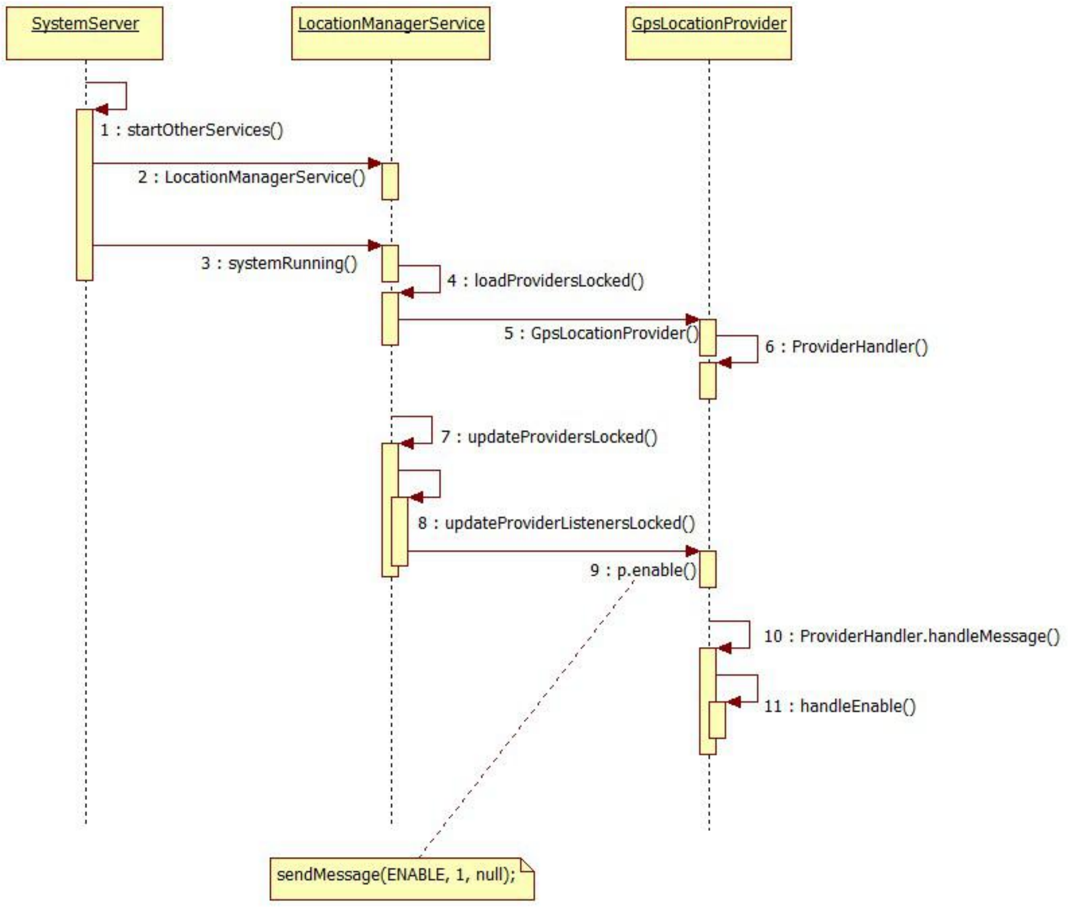
目录

QCOM GPS init

GPS 初始化流程，主要介绍 LMS 是怎样创建的，FW 是怎样拿到 HAL 提供的 GPS 相关接口的。

1 启动 LocationManagerService

和其他系统 service 一样， LocationManagerService 在 SystemServer 中创建， LMS 调用 systemRunning 来加载和设置各种 location provider，这里只介绍 GPS provider enable 的过程。


SystemServer
LocationManagerService
GpsLocationProvider
1 : startOtherServices()
2 : LocationManagerService()
3 : systemRunning()
4 : loadProvidersLocked()
5 : GpsLocationProvider()
6 : ProviderHandler()
7 : updateProvidersLocked()
8 : updateProviderListenersLocked()
9 : p.enable()
10 : ProviderHandler.handleMessage()
11 : handleEnable()
sendMessage(ENABLE, 1, null);

frameworks/base/services/java/com/android/server/SystemServer.java

```
private void startOtherServices() {
    ... ...
        Slog.i(TAG, "Location Manager");
        // 创建 LMS 对象
        location = new LocationManagerService(context);
        ServiceManager.addService(Context.LOCATION_SERVICE, location);
    ... ...
    final LocationManagerService locationF = location;
    ... ...
     try {
        //创建的 LMS 对象不为空时，调用 systemRunning 启动 LMS 服务
        if (locationF != null) locationF.systemRunning();
     } catch (Throwable e) {
        reportWtf("Notifying Location Service running", e);
    }
    ... ...
}
```

frameworks/base/services/core/java/com/android/server/LocationManagerService.java

```
    public void systemRunning() {
        synchronized (mLock) {
            ... ...

            // prepare providers
            /*加载所有可以提供位置信息的 provider，例如 GpsLocationProvider
            networkProvider */
            loadProvidersLocked();
            //根据设置 enable or disable loadProvidersLocked 中加载的 provider
            updateProvidersLocked();
            ... ...
        }
     }

    private void loadProvidersLocked() {
        // create a passive location provider, which is always enabled
        ... ...
        //如果支持 GpsLocationProvider，则创建一个 GpsLocationProvider 对象
        if (GpsLocationProvider.isSupported()) {
            // Create a gps location provider
            GpsLocationProvider gpsProvider = new GpsLocationProvider(mContext, this,
                    mLocationHandler.getLooper());
            ... ...
        }
```

frameworks/base/services/core/java/com/android/server/location/GpsLocationProvider.java

```
        public GpsLocationProvider(Context context, ILocationManager ilocationManager,
                Looper looper) {
          ... ...

            // Construct internal handler
            //创建一个 Handler，用处理 enable/disable GpsLocationProvider 等消息
            mHandler = new ProviderHandler(looper);
          ... ...
        }

        public ProviderHandler(Looper looper) {
            super(looper, null, true /*async*/);
        }

        @Override
        public void handleMessage(Message msg) {
            int message = msg.what;
            switch (message) {
                 case ENABLE:
                     if (msg.arg1 == 1) {
                         //收到 enable 消息时，调用 handleEnable 方法
                         handleEnable();
        ... ...
        }
```

frameworks/base/services/core/java/com/android/server/LocationManagerService.java

```
            private void updateProvidersLocked() {
                boolean changesMade = false;
                for (int i = mProviders.size() - 1; i >= 0; i--) {
                    LocationProviderInterface p = mProviders.get(i);
                    ... ...
                    if (isEnabled && !shouldBeEnabled) {
                        updateProviderListenersLocked(name, false);
            ... ...
            }
            private void updateProviderListenersLocked(String provider, boolean enabled) {
                int listeners = 0;
                /*所有提供位置信息的 provider 都要实现接口 LocationProviderInterface */
```

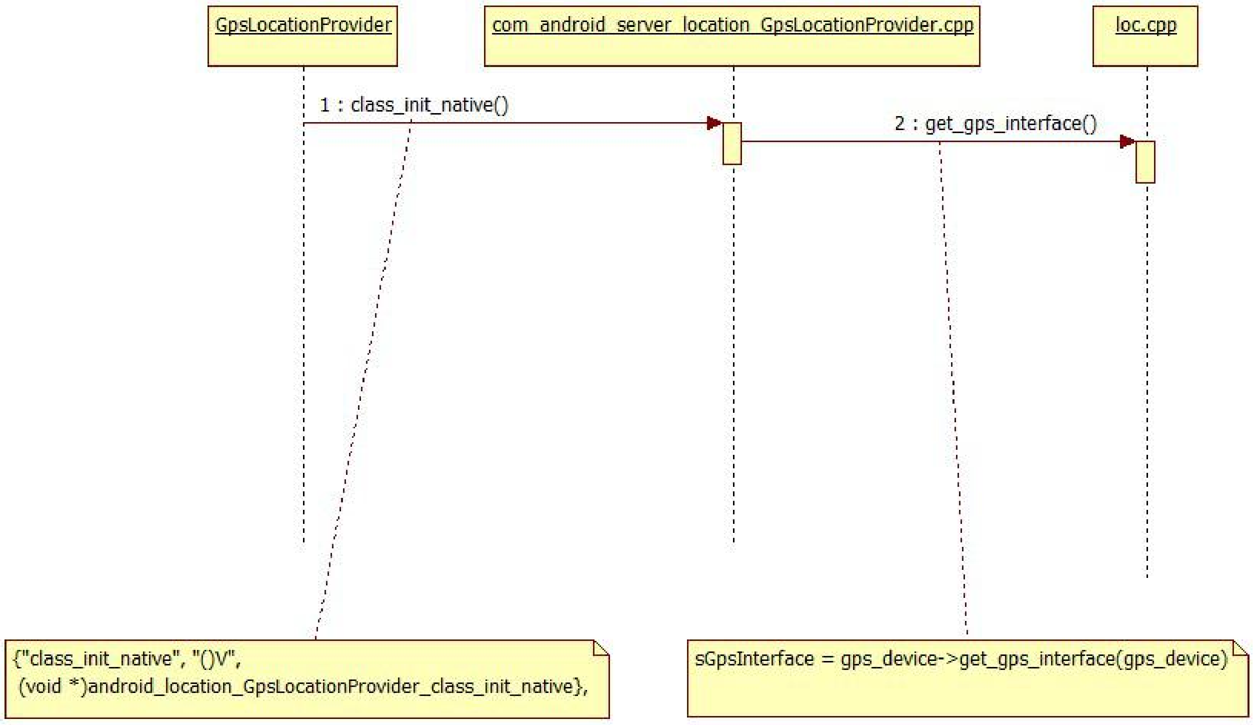
```
        LocationProviderInterface p = mProvidersByName.get(provider);
        ... ...

        if (enabled) {
            /*由于 GpsLocationProvider 实现了实现接口 LocationProviderInterface，
            因此这里调用的是 GpsLocationProvider 的方法*/
            p.enable();
            if (listeners > 0) {
                applyRequirementsLocked(provider);
            }
        } else {
            p.disable();
        }
    }
```

frameworks/base/services/core/java/com/android/server/location/GpsLocationProvider.java

```
    public void enable() {
        synchronized (mLock) {
            if (mEnabled) return;
            mEnabled = true;
        }
        /*发送 enable 消息，由 ProviderHandler 接收并处理，
        最终调用 handleEnable 方法*/
        sendMessage(ENABLE, 1, null);
    }
```

2 获得 GpsInterface

GpsLocationProvider 是怎么样 enable 的，在上一节中已经介绍了。这部分主要接收 GpsLocationProvider 怎样拿到 Jni 层提供的 Gps 接口(这里的 Gps 接口指的是 enable/disable GPS，report location， 解析 nmea 数据等接口)。


GpsLocationProvider
com_android_server_location_GpsLocationProvider.cpp
loc.cpp
1 : class_init_native()
2 : get_gps_interface()
{"class_init_native", "()V",
(void *)android_location_GpsLocationProvider_class_init_native},
sGpsInterface = gps_device->get_gps_interface(gps_device)

frameworks/base/services/core/java/com/android/server/location/GpsLocationProvider.java

```
public class GpsLocationProvider implements LocationProviderInterface {
    ... ...
    // 当创建 GpsLocationProvider 对象时，就会调用 class_init_native
      static { class_init_native(); }
      ... ...
}
```

frameworks/base/services/core/jni/com_android_server_location_GpsLocationProvider.cpp

```
// class_init_native 实际上调用的是 android_location_GpsLocationProvider_class_init_native
{"class_init_native", "()V", (void *)android_location_GpsLocationProvider_class_init_native},

static void android_location_GpsLocationProvider_class_init_native(JNIEnv* env, jclassclazz) {
     ... ...
err = hw_get_module(GPS_HARDWARE_MODULE_ID, (hw_module_tconst**)&module);
      if (err == 0) {
            hw_device_t* device;
            //打开 id 为 GPS_HARDWARE_MODULE_ID 的 lib，调用 lib 的 open 方法
            err = module->methods->open(module, GPS_HARDWARE_MODULE_ID, &device);
            if (err == 0) {
                  gps_device_t* gps_device = (gps_device_t *)device;
                  /*得到 gps 接口，实际上就是 loc.cpp 中的一个数组 sLocEngInterface，
                  详细分析过程参考后面的代码，这里只需要记住通过 sGpsInterface 调用的
```

```
                    都是 sLocEngInterface 里面的函数*/
                    sGpsInterface = gps_device->get_gps_interface(gps_device);
            }
        }
}
```

android/hardware/qcom/gps/loc_api/libloc_api_50001/gps.c

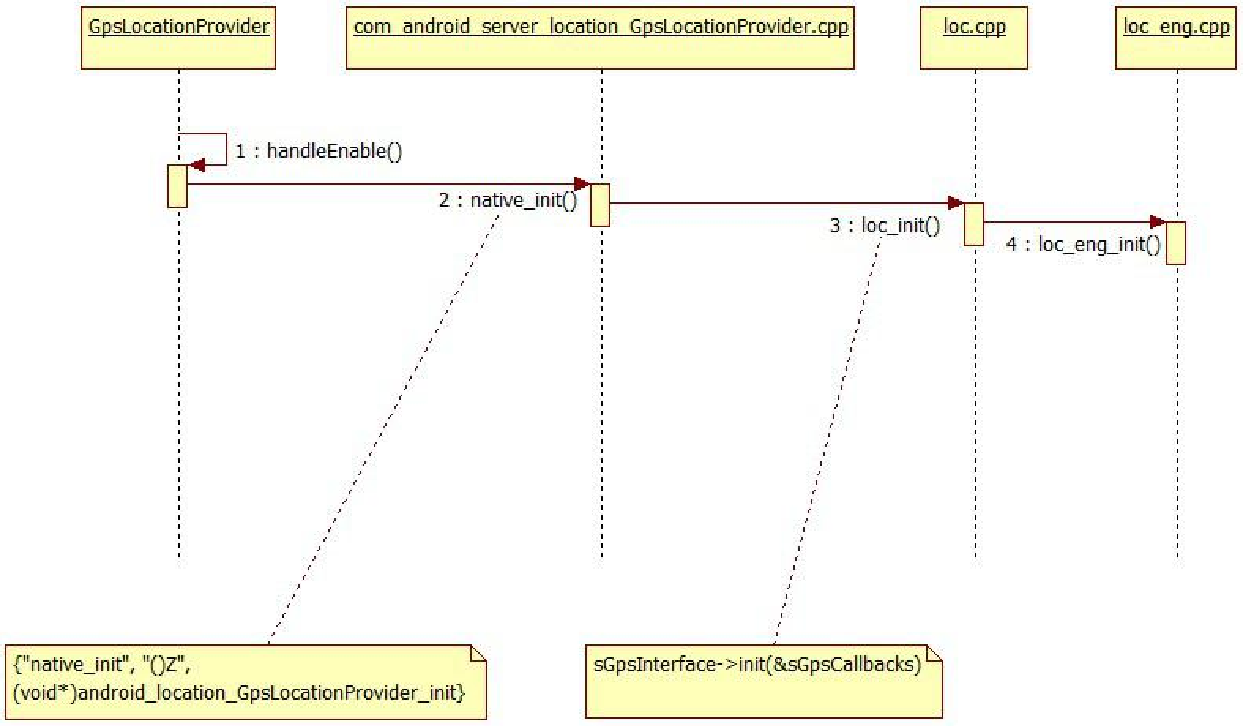
```
struct hw_module_t HAL_MODULE_INFO_SYM = {
        .tag = HARDWARE_MODULE_TAG,
        .module_api_version = 1,
        .hal_api_version = 0,
        //通过 id= GPS_HARDWARE_MODULE_ID，定位到 open 的哪一个 lib
        .id = GPS_HARDWARE_MODULE_ID,
        .name = "loc_api GPS Module",
        .author = "Qualcomm USA, Inc.",
        // methods 调用的是 gps_module_methods 函数
        .methods = &gps_module_methods,
};

static struct hw_module_methods_t gps_module_methods = {
        //实际上 open 调用的是 open_gps 函数
        .open = open_gps
};


static int open_gps(const struct hw_module_t* module, char const* name,
                struct hw_device_t** device)
{
        struct gps_device_t *dev = (struct gps_device_t *) malloc(sizeof(struct gps_device_t));

        if(dev == NULL)
                return -1;

        memset(dev, 0, sizeof(*dev));

        dev->common.tag = HARDWARE_DEVICE_TAG;
        dev->common.version = 0;
        dev->common.module = (struct hw_module_t*)module;
        /* android_location_GpsLocationProvider_class_init_native 中的
        sGpsInterface 实际上就是 gps__get_gps_interface*/
        dev->get_gps_interface = gps__get_gps_interface;
        *device = (struct hw_device_t*)dev;
        return 0;
}
```

```
const GpsInterface* gps__get_gps_interface(struct gps_device_t* dev)
{
    //GPS 接口由 loc.cpp 中的 get_gps_interface 函数返回
    return get_gps_interface();
}
```

android/hardware/qcom/gps/loc_api/libloc_api_50001/loc.cpp

```
extern "C" const GpsInterface* get_gps_interface()
{
    ... ...
    // get_gps_interface 返回的其实就是一个数组
   return &sLocEngInterface;
}

static const GpsInterface sLocEngInterface =
{
   sizeof(GpsInterface),
   loc_init,
   loc_start,
   loc_stop,
   loc_cleanup,
   loc_inject_time,
   loc_inject_location,
   loc_delete_aiding_data,
   loc_set_position_mode,
   loc_get_extension
};
```

3 初始化


GpsLocationProvider
com_android_server_location_GpsLocationProvider.cpp
loc.cpp
loc_eng.cpp
1 : handleEnable()
2 : native_init()
3 : loc_init()
4 : loc_eng_init()
{"native_init", "()Z", (void*)android_location_GpsLocationProvider_init}
sGpsInterface->init(&sGpsCallbacks)

这里的初始化指的是 native_init，主要是把 JNI 层设置的 sGpsCallbacks 接口通过 init 函数传给 hal 层。

frameworks/base/services/core/java/com/android/server/location/GpsLocationProvider.java

```
    private final class ProviderHandler extends Handler {
            ... ...
        @Override
        public void handleMessage(Message msg) {
            int message = msg.what;
            switch (message) {
                case ENABLE:
                    if (msg.arg1 == 1) {
                        handleEnable();
                    } else {
                        handleDisable();
                    }
                    break;
          ... ...
    }

  private void handleEnable() {
        if (DEBUG) Log.d(TAG, "handleEnable");
        //native_init 是 GPS provider enable 时调用到的
        boolean enabled = native_init();
        ... ...
 }
```

frameworks/base/services/core/jni/com_android_server_location_GpsLocationProvider.cpp

```
{"native_init", "()Z", (void*)android_location_GpsLocationProvider_init},

static jboolean android_location_GpsLocationProvider_init(JNIEnv* env, jobject obj)
{
    ... ...
// fail if the main interface fails to initialize
//还记得吗？这个 sGpsInterface 就是 sLocEngInterface， 相当于这里的 init 调用的是 loc_init
if (!sGpsInterface || sGpsInterface->init(&sGpsCallbacks) != 0)
        return JNI_FALSE;

    ... ...

    return JNI_TRUE;
}

GpsCallbacks sGpsCallbacks = {
   sizeof(GpsCallbacks),
   location_callback,
   status_callback,
   sv_status_callback,
   nmea_callback,
   set_capabilities_callback,
   acquire_wakelock_callback,
   release_wakelock_callback,
  create_thread_callback,
   request_utc_time_callback,
};

static pthread_t create_thread_callback(const char* name, void (*start)(void *), void* arg)
{
   return (pthread_t)AndroidRuntime::createJavaThread(name, start, arg);
}
```

下面就是 create_thread_callback 往下传的一个调用流程，直接看代码，这里不再描述了。

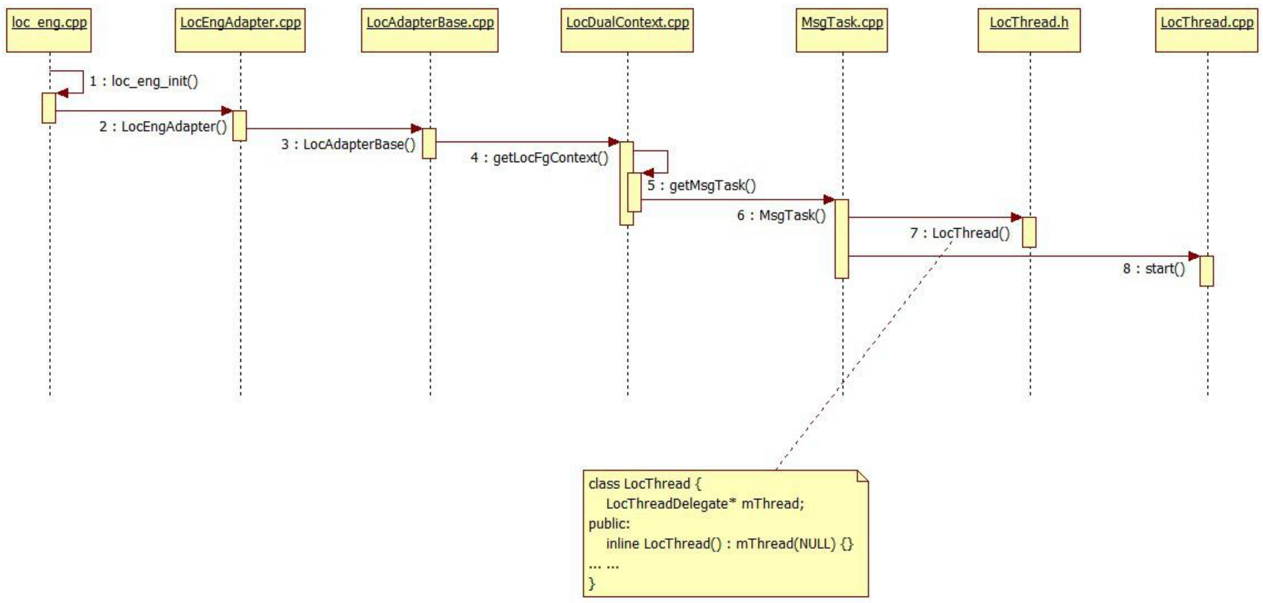
android/hardware/qcom/gps/loc_api/libloc_api_50001/loc.cpp

```
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 */
                                        callbacks->acquire_wakelock_cb, /* acquire_wakelock_cb */
                                        callbacks->release_wakelock_cb, /* release_wakelock_cb */
                                        callbacks->create_thread_cb, /* create_thread_cb */
                                        NULL, /* location_ext_parser */
                                        NULL, /* sv_ext_parser */
                                        callbacks->request_utc_time_cb, /* request_utc_time_cb */
                                        };
    gps_loc_cb = callbacks->location_cb;
    gps_sv_cb = callbacks->sv_status_cb;
    retVal = loc_eng_init(loc_afw_data, &clientCallbacks, event, NULL);
    ... ...
}
```

hardware/qcom/gps/loc_api/libloc_api_50001/loc_eng.cpp

```
int loc_eng_init(loc_eng_data_s_type &loc_eng_data, LocCallbacks* callbacks,
                 LOC_API_ADAPTER_EVENT_MASK_Tevent, ContextBase* context)

{
    ... ...
// Save callbacks

    loc_eng_data.location_cb = callbacks->location_cb;
    ... ...

    loc_eng_data.adapter =
        new LocEngAdapter(event, &loc_eng_data, context,
                          (LocThread::tCreate)callbacks->create_thread_cb);
    ... ...
    loc_eng_data.adapter->sendMsg(new LocEngInit(&loc_eng_data));
    EXIT_LOG(%d, ret_val);
    return ret_val;
}
```

4 创建 thread



```
class LocThread {
    LocThreadDelegate* mThread;
public:
    inline LocThread() : mThread(NULL) {}
... ...
}
```

在 native_init 初始化的过程中还会创建一个 thread，在 GPS init 阶段还不出来这个 thread 到底有什么作用，在 GPS start 中会讲到，这个 tread 是用来处理 hal 层 start gps 等消息的。

hardware/qcom/gps/loc_api/libloc_api_50001/LocEngAdapter.cpp

//知道怎么调到 LocEngAdapter 的吗？不知道就再回去看看 loc_init 里面的调用

/*C++代码，很多关键的调用都"藏"成员变量的初始化里面，找不到调用的时候，一定要仔细构造方法里面的成员变量初始化*/

```
LocEngAdapter::LocEngAdapter(LOC_API_ADAPTER_EVENT_MASK_T mask,
                             void* owner, ContextBase* context,
                             LocThread::tCreate tCreator) :
   LocAdapterBase(mask,
                  //Get the AFW context if VzW context has not already been intialized in
                  //loc_ext
                  context == NULL?
                  LocDualContext::getLocFgContext(tCreator,
                                                  NULL,
                                                  LocDualContext::mLocationHalName,
                                                  false)
                  :context),
    mOwner(owner), mInternalAdapter(new LocInternalAdapter(this)),
    mUlp(new UlpProxyBase()), mNavigating(false),
    mSupportsAgpsRequests(false),
    mSupportsPositionInjection(false),
    mSupportsTimeInjection(false),
    mPowerVote(0)
{
    memset(&mFixCriteria, 0, sizeof(mFixCriteria));
    mFixCriteria.mode = LOC_POSITION_MODE_INVALID;
    LOC_LOGD("LocEngAdapter created");
}
```

hardware/qcom/gps/core/LocDualContext.cpp

```
ContextBase* LocDualContext::getLocFgContext(LocThread::tCreate tCreator,
            LocMsg* firstMsg, const char* name, bool joinable)
{
    pthread_mutex_lock(&LocDualContext::mGetLocContextMutex);
    LOC_LOGD("%s:%d]: querying ContextBase with tCreator", __func__, __LINE__);
    if (NULL == mFgContext) {
        LOC_LOGD("%s:%d]: creating msgTask with tCreator", __func__, __LINE__);
        //获得 MsgTask
        const MsgTask* msgTask = getMsgTask(tCreator, name, joinable);
        mFgContext = new LocDualContext(msgTask,
                                        mFgExclMask);
    }
    ... ...
}

const MsgTask* LocDualContext::getMsgTask(LocThread::tCreate tCreator,
                                          const char* name, bool joinable)
{
    if (NULL == mMsgTask) {
       //创建 MsgTask 对象
       mMsgTask = new MsgTask(tCreator, name, joinable);
    }
    return mMsgTask;
}
```

hardware/qcom/gps/utils/MsgTask.cpp

```
MsgTask::MsgTask(LocThread::tCreate tCreator,
                 const char* threadName, bool joinable) :
    //mThread 就是一个 LocThread 对象
    mQ(msg_q_init2()), mThread(new LocThread()) {
    //调用 LocThread 的 start 函数，其实上就是创建了一个 LocThreadDelegate 对象
    if (!mThread->start(tCreator, threadName, this, joinable)) {
        delete mThread;
        mThread = NULL;
    }
}
```

hardware/qcom/gps/utils/LocThread.h

```
class LocThread {
    LocThreadDelegate* mThread;
public:
    inline LocThread() : mThread(NULL) {}
... ...
}
```

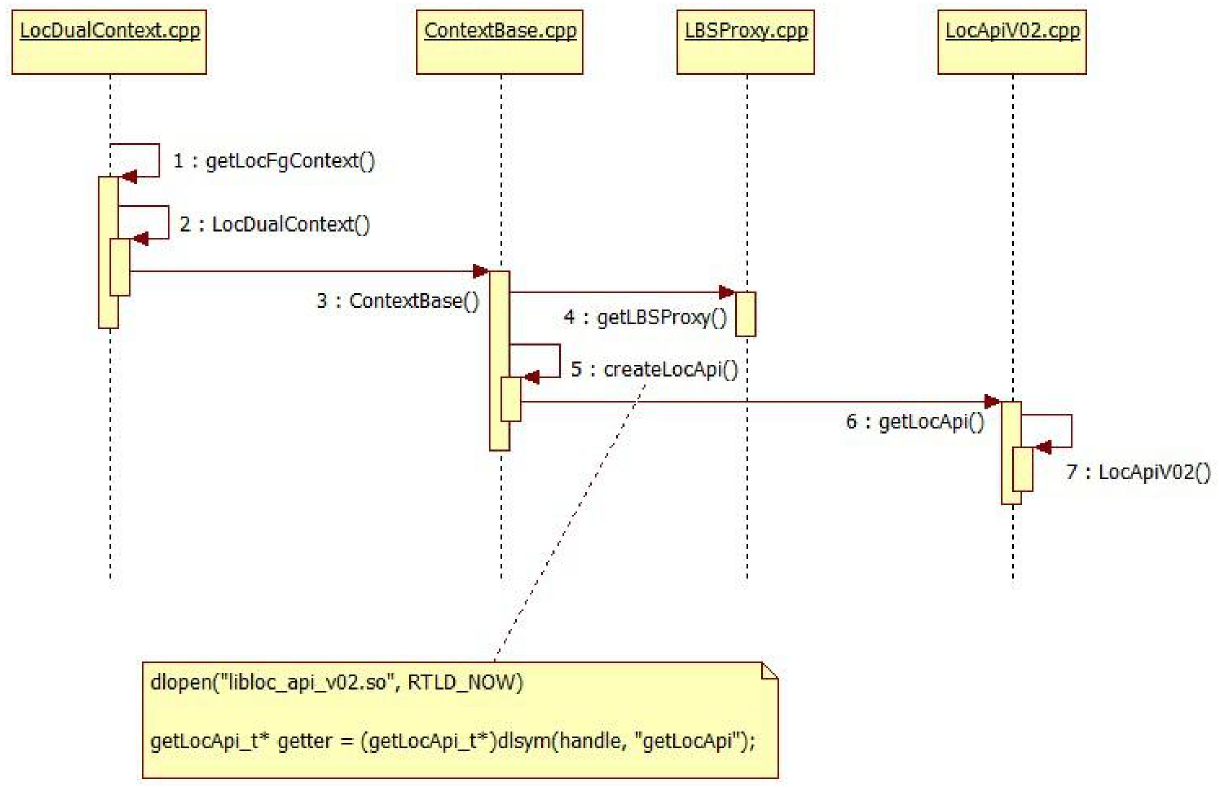
hardware/qcom/gps/utils/LocThread.cpp

```
bool LocThread::start(tCreate creator, const char* threadName, LocRunnable* runnable, bool
joinable) {
    bool success = false;
    if (!mThread) {
        mThread = LocThreadDelegate::create(creator, threadName, runnable, joinable);
        // true only if thread is created successfully
        success = (NULL != mThread);
    }
    return success;
}


LocThreadDelegate* LocThreadDelegate::create(LocThread::tCreate creator,
        const char* threadName, LocRunnable* runnable, bool joinable) {
    LocThreadDelegate* thread = NULL;
    if (runnable) {
        thread = new LocThreadDelegate(creator, threadName, runnable, joinable);
        if (thread && !thread->isRunning()) {
            thread->destroy();
            thread = NULL;
        }
    }
    return thread;
}
```

5 获得 LocApiV02

重点来了，前面说了这么多，还是没看到调用的 GPS 哪个.so 提供的。这部分会说明 gps 用

到接口都是由 libloc_api_v02.so 提供的。



hardware/qcom/gps/core/LocDualContext.cpp

// getLocFgContext 调用参考创建 thread 里面的调用流程

```
ContextBase* LocDualContext::getLocFgContext(LocThread::tCreate tCreator,
            LocMsg* firstMsg, const char* name, bool joinable)
{
    ... ...
        //创建 LocDualContext 对象
        mFgContext = new LocDualContext(msgTask,
                                        mFgExclMask);
    }
   ... ...
}

LocDualContext::LocDualContext(const MsgTask* msgTask,
                               LOC_API_ADAPTER_EVENT_MASK_T exMask) :
   ContextBase(msgTask, exMask, mLBSLibName)
{
}
```

hardware/qcom/gps/core/ContextBase.cpp

```
ContextBase::ContextBase(const MsgTask* msgTask,
                         LOC_API_ADAPTER_EVENT_MASK_T exMask,
                         const char* libName) :
    mLBSProxy(getLBSProxy(libName)),
    mMsgTask(msgTask),
    //调用 createLocApi
    mLocApi(createLocApi(exMask)),
    mLocApiProxy(mLocApi->getLocApiProxy())
{
}
}

LocApiBase* ContextBase::createLocApi(LOC_API_ADAPTER_EVENT_MASK_T exMask)
{
    LocApiBase* locApi = NULL;
    ... ...
          //打开 libloc_api_v02.so
          if((handle = dlopen("libloc_api_v02.so", RTLD_NOW)) != NULL) {
              LOC_LOGD("%s:%d]: libloc_api_v02.so is present", __func__, __LINE__);
              //获得 libloc_api_v02.so 中的 getLocApi 函数
              getLocApi_t* getter = (getLocApi_t*)dlsym(handle, "getLocApi");
              if(getter != NULL) {
                  LOC_LOGD("%s:%d]: getter is not NULL for LocApiV02", __func__, __LINE__);
                  locApi = (*getter)(mMsgTask, exMask, this);
              }
          }
    ... ...
}
```