怎么写好 sensorHAL

展讯带你破解传感器迷题

1. 背景介绍

SensorHAL 是传感器模块中上接 framework 下连驱动的重要一层，需要对所有 sensor 有一定的了解才能写好，为了能使得在展讯平台上编写调试 sensorHAL 的广大用户能更轻松的完成这个任务，制作本文档

2. 调通，读懂你的 sensor 驱动

在 google 的定义中，驱动层采用的是各个器件厂商的标准 linux 驱动代码，而 framework 是 android 的标准接口。HAL 才是真正的 porting 层，不变的 framework 遇到千变的驱动，首先读懂驱动代码才是写好 sensorHAL 的第一步。

2.1 如何移植 sensor 驱动

2.1.1 怎样把对应的 sensor 驱动编入版本

sensor 厂商的驱动一般通过 ko（内核模块）的方式提供，在展讯平台上，驱动代码的移植有一定的规则

首先请按照 sensor 的类别统一放置驱动代码在对应目录

3rdparty/gsensor（重力传感器）

3rdparty/msensor（磁传感器）

3rdparty/psensor（接近传感器）

3rdparty/lsensor（光传感器）

如果是多合一的芯片，可以任意放在其中一个目录下。

其次，需要在 mak 文件中指定对应的 sensor，例如在 customize/make/sp6820a.mak 中对 sensor 的编译选择如下

3RDPARTY_GSENSOR = lis3dh

3RDPARTY_MSENSOR = akm8975

3RDPARTY_LSENSOR = AL3006

假设 gsensor 改成了 ADXL346，只需要把 gsensor 这行改为

3RDPARTY_GSENSOR = ADXL346

最后，需要保证驱动代码所在目录下的几个关键文件符合展讯的规则，我们就以 lis3dh 这颗 gsensor 为例做说明

进入 3rdparty/gsensor/lis3dh/special 目录，可以所有的文件都放在这个目录下，special 目录就是编译系统识别驱动的入口

build.sh 是编译脚本，在移植自己的 sensor 驱动时基本上可以直接拷贝这个文件。需要注意的是，必须保证这个文件拥有可执行权限，否则驱动不能参与编译

init.3rdparty.rc 指定了驱动添加后需要增加的 rc 脚本项，在 lis3dh 的驱动中，这个文件包含两行

on boot

    insmod /system/lib/modules/lis3dh_acc.ko   //挂载 gsensor 的驱动

on init

device /dev/lis3dh_acc 660 system system //设定驱动设备文件节点的权限

driver/目录下是具体驱动文件

driver/Makefile 指定了参与编译的驱动文件

2.1.2 怎样单独编译调试 ko

可以使用

./mk $(projname) u ko $(3rdpath)指令进行编译

例如

./mk sp6820a u ko 3rdparty/gsensor/lis3dh

生成的 ko 可以通过 adb 推送和调试

2.1.3 一般移植中会产生哪些代码修改

一般会需要重新配置相关的 pinmap，gpio 和 i2c port。这些配置的方法，请参考《展讯平台驱动配置文档》

2.2 如何理解我的 sensor 驱动

做 HAL 层之前我们必须先看懂 sensor 驱动，这是编写 HAL 的前提

2.2.1 sensor 驱动需要给 HAL 提供什么

需要提供 input 设备并通过这个设备上报事件

所以我们在所有的 sensor 驱动中都能看到 input_register_device 这样的 API，这就是注册 input 设备的地方，你需要记住注册时 sensor 驱动为自己的 input 设备起的名字，这是 HAL 层需要识别的标志之一

需要特别注意的是，sensor 可以实现一个或者多个 input 设备，或者几个 sensor 共用一个 input 设备。所以物理上 sensor 的个数，逻辑上 sensor 的个数，和 sensor 驱动实现的 input 设备数是三个不同的概念

需要提供 enable delay 等接口

提供上层控制接口的方式是多种多样的，目前比较常见的有两种方式，ioctl 方式和 sysfs 的方式。

ioctl 方式是一种传统的 linux 控制设备节点的方式，常见的会注册一个 misc 设备（使用 misc_register 接口，不同于上报事件的 input 设备），并提供一组 IOCTL 的定义，HAL 层需要识别 misc 设备的设备名和 IOCTL 的定义（lis3dh）

sysfs 方式相对较晚出现，这种方式通过实现一组 device_attribute 来提供面向用户空间的接口,可以有多种方式，实现后会在手机 sys 目录下生成对应的文件节点，对文件进行读写就会调用到驱动中对应的方法。(参考 akm8975)

2.3 如何确定我的驱动已经 work

2.3.1 查看对应的节点是否已经生成

通过 adb shell，首先查看 input 设备是否成功注册，这可以直接通过 getevent，这条指令会枚举所有注册成功的 input 设备。对于 ioctl 方式，还需要查看 dev 目录下对应的 misc 设备是否存在，sysfs 的方

式

需要查看 sys/目录对应的目录下文件是否生成，比如 akm8975，就会生成 sys/class/compass/akm8975 目录，并在该目录下生成 mag_enable,mag_delay 等文件节点

2.3.2 确认 enable delay 接口有效

ioctl 方式，可以编写一小段测试代码（可以参考 lis3dh_test.c），调用 ioctl。

sysfs 方式，可以直接 adb shell 后通过 cat echo 相关文件节点的方式确认工作

例如

echo 1 > /sys/class/compass/akm8975/mag_enable

2.3.3 确认驱动上报事件

在 enable 相关驱动后，通过 getevent 指令观察是否有事件上报

3. sensorHAL 之 2.2.2 篇

3.1 修改哪个文件

rdparty/app/app6820/special/android/hardware/sprd/hsdroid/libsensors/sprd_sensors_sp6820a.c

3.2 起手

struct sensors_module_t HAL_MODULE_INFO_SYM

这个结构体的定义是 senseorHAL 的起点。这里有两个方法是值得注意的

__get_sensors_list 向 framework 提供支持的 sensor 的基本信息，最终返回的是 static struct sensor_t sensors_list[SENSORS_SUPPORT_COUNT]这个静态数组。第一步需要填写这个数组

__module_methods 结构，最终是结构的成员，__module_methods_open，这是 sensorHAL 加载后会被调用的第一个函数。这个函数负责向上层注册八个回调函数，这八个函数又分为控制流的 5 个和数据流的 3 个。将完成 sensorHAL 的所有功能

3.3 八回调之控制流

3.3.1 control__open_data_source

这是最重要的一个回调，它将打开找到所有 sensor 驱动注册的 input 设备并且将这些设备节电传递给数据流的回调。这是通过返回 native_handle_t 这个结构提实现的。data 成员中包含了所有 input 设备句柄

需要注意修改的有几点

SENSOR_DEVICE_COUNT 指的不是 sensor 的个数，而是 input 设备的个数，这个值必须配置正确

control__open_data_source 调用的 open_inputs 函数内部需要修改，

确保 fd_inputs 数组的每一个成员都被赋予合法的值，否则 framework 会产生一个错误，注意正确配置驱动中注册 input 设备名，我们通过几个宏来配置这些 input 设备的名字

#define SENSOR_ACC_INPUT_NAME "accelerometer"

#define SENSOR_MAG_INPUT_NAME "compass"

#define SENSOR_PXL_INPUT_NAME "proximity"

SENSOR_DEVICE_ACC SENSOR_DEVICE_MAG SENSOR_DEVICE_PXL 等 ID 需要连续，比如没有 Msensor 的情况，SENSOR_DEVICE_PXL 需要代替 SENSOR_DEVICE_MAG 配置成 1，又比如 psensor 和 lsensor 是两个 input 设备的情况，需要修改这边的定义，将 SENSOR_DEVICE_PXL 拆分为 SENSOR_DEVICE_P 和 SENSOR_DEVICE_L 等

3.3.2 control__close

这个回调为所有 sensor 提供一个自己清理自己资源的方法，在 2.2.2 默认的 sensorHAL 中，对 misc 设备做了关闭的动作。根据自己的需要实现这个函数，也可能是什么都不做

3.3.3 control__activate

调用各个 sensor 驱动的 enanle 接口，ioctl 方式实现的请参照 control_enable_disable_acc_sensor 中对 lis3dh 驱动的操作方式，需要的驱动信息是 misc 设备名和 ioctl 的定义。我们也有几个宏来定义了 misc 设备的名字

#define SENSOR_ACC_DEVICE_NAME "/dev/lis3dh_acc"

#define SENSOR_MAG_DEVICE_NAME "/dev/akm8975_dev"

#define SENSOR_PXL_DEVICE_NAME "/dev/al3006_pls"

sysfs 方式的实现参照 control_enable_disable_mag_sensor 对 akm8975 驱动，需要的驱动信息是 sysfs 文件名和读写的值

3.3.4 control__set_delay

调用各个 sensor 驱动 delay 接口（有些 sensor 可能没有）。方法同 control__activate

3.3.5 control__wake

一般不需要修改，只要已经保证 open_inputs 正确即可

3.4 八回调之数据流

3.4.1 data__data_open

这个函数一般可以不用修改，其实是通过 dup 方法将 control__open_data_source 时保存的 input 设备句柄取出来

3.4.2 data__close

这个函数一般可以不用修改，完成关闭 input 设备的工作

3.4.3 data__poll

这个函数中实现各个 sensor 获得数据的方式，按 sensor 类型一一介绍

gsensor

参考 data__poll_acceleration_sensor

type 必须是 EV_ABS（绝对值），支持的 code 包括 EVENT_TYPE_ACCEL_X，EVENT_TYPE_ACCEL_Y EVENT_TYPE_ACCEL_Z，对应的 value 分别填入 dev 参数的 sensors[ID_A].acceleration.x sensors[ID_A].acceleration.y sensors[ID_A].acceleration.z 三个成员 value 必须通过 CONVERT 常数来转换，这个值指是一个 G（重力加速度）相对的上报值

msensor

参考 data__poll_magnetic_sensor

type 必须是 EV_ABS（绝对值），支持的 code 包括 EVENT_TYPE_MAGV_X EVENT_TYPE_MAGV_Y EVENT_TYPE_MAGV_Z 对应的 value 填入 dev 参数的 dev->sensors[ID_M].magnetic.x dev->sensors[ID_M].magnetic.y dev->sensors[ID_M].magnetic.z 成员 value 也要转换，转换的常数是一个特丝拉对应的上报值

osensor

同样参考 data__poll_magnetic_sensor（这个其实由 msensor 厂家提供）

type 必须是 EV_ABS（绝对值），支持的 code 包括 EVENT_TYPE_ORIENT_YAW EVENT_TYPE_ORIENT_PITCH

EVENT_TYPE_ORIENT_ROLL 对应的 value 分别填入 dev 参数的 sensors[ID_O].orientation.azimuth sensors[ID_O].orientation.pitch sensors[ID_O].orientation.roll 三个成员 转换常数由 sansor 厂商定义，实际为 1 个弧度对应的值

psensor

参考 data__poll_proximity_sensor

type 同样是 EV_ABS（绝对值），支持 code EVENT_TYPE_PROXIMITY，value 为 1 或者 0，填入 dev 参数的 sensors[ID_P].distance 成员

lsensor

参考 data__poll_proximity_sensor

type 同样是 EV_ABS（绝对值），支持 code EVENT_TYPE_LIGHT，value 为上报光强度，填入 dev 参数的 sensors[ID_L].light 成员

4. sensorHAL 之 2.3.5 篇

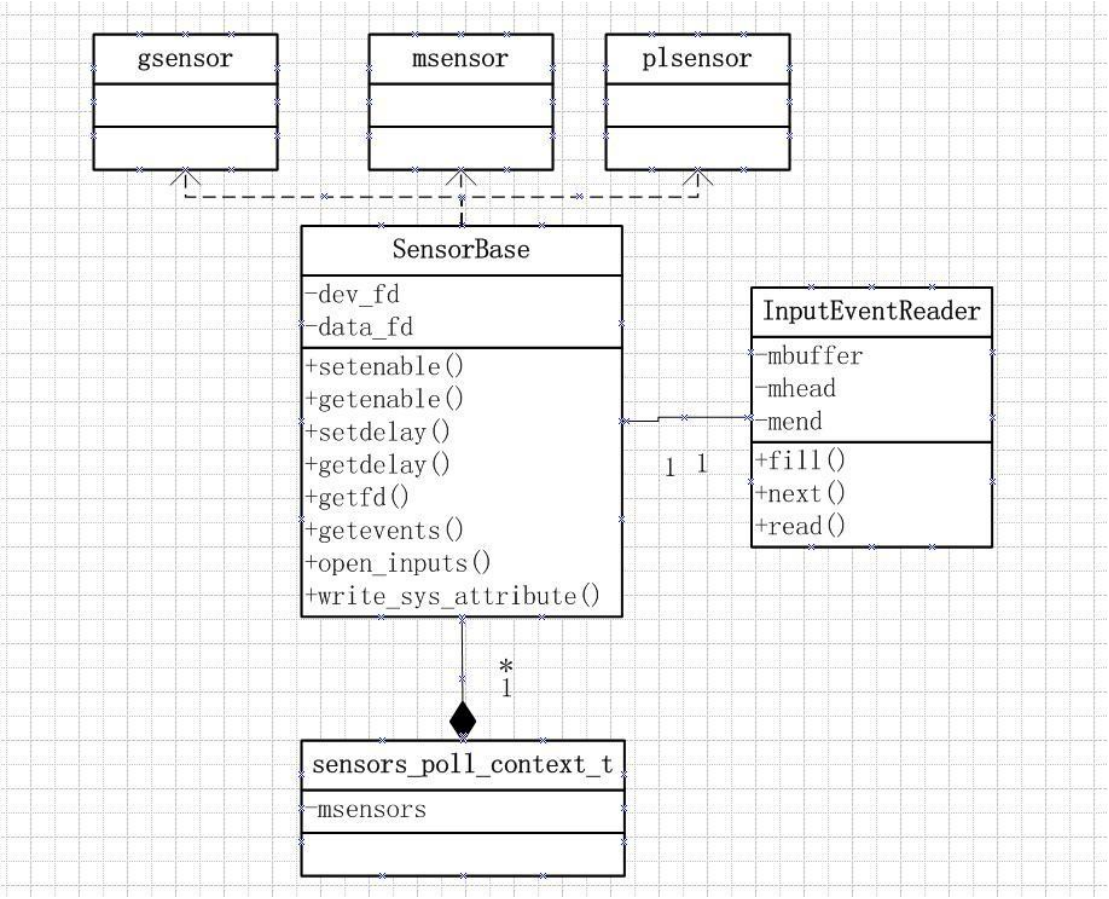
4.1 修改哪些文件

3rdparty/app/app8810/special/android/hardware/sprd/hsdroid/libsensors/下增加自己的 sensorHAL 实现

4.2 有了对象之后

汇编，c 和 c++是好兄弟，以前经常一起吃饭，后来 c++不来了，因为人家有了对象。

在 2.3.5 中对所有 sensor 在 HAL 中的实现抽象出了一个基类，SensorBase（在 SensorBase.cpp 中实现），同时对所有 sensor 需要的 input 处理封装成了另一个类 InputEventReader，所有 sensor 驱动都继承 SensorBase，同时实现一个 InputEventReader 的实例，并且所有的 sensor 都有 sensors.cpp 中的结构体 sensors_poll_context_t 来管理



4.3 实现自己的 sensor 子类

还是拿 lis3dh 为例，需要从基类派生

class Lis3dhSensor : public SensorBase

重要的成员包括

int mEnabled; //保存 enable 状态

int64_t mDelay; //保存 delay 状态

InputEventCircularReader mInputReader; //InputEventReader 的实现

sensors_event_t mPendingEvent; //保存未处理的事件

bool mHasPendingEvent; //是否有未处理的事件

char input_sysfs_path[PATH_MAX]; //sysfs 接口的路径

int input_sysfs_path_len; //sysfs 接口路径的长度

需要重写的函数包括

Lis3dhSensor(); //构造函数

virtual ~Lis3dhSensor(); //析构函数

virtual int readEvents(sensors_event_t* data, int count); //读事件的方法

virtual bool hasPendingEvents() const; //是否有未处理的事件

virtual int setDelay(int32_t handle, int64_t ns); //delay 接口

virtual int setEnable(int32_t handle, int enabled); //enable 接口

virtual int64_t getDelay(int32_t handle); //delay 接口

virtual int getEnable(int32_t handle); //读取 enable 接口

4.3.1 构造函数与析构函数

首先调用父类构造函数 SensorBase(LIS3DH_ACC_DEV_PATH_NAME, LIS3DH_ACC_INPUT_NAME)两个参数分别是 misc 设备的设备名和 input 设备的设备名。因为 lis3dh 使用 ioctl 方式，所以有 misc 设备，

如果没有使用 misc 设备，则第一个参数给 NULL 就可以。然后初始化各个重要参数，最后通过 open_device 来打开 misc 设备备用

如果是 sysfs 方式的驱动，HAL 还需要设定 input_sysfs_path

析构函数主要是关掉 enable 和 close_device

4.3.2 enable 接口和 delay 接口

对于 ioctl 方式，向基类 dev_fd 成员发送 ioctl，对于 sysfs，调用基类方法 write_sys_attribute

4.3.3 实现 readEvents

这是读取事件的函数，流程是首先 mInputReader.fill，然后循环调用 mInputReader.readEvent 和 mInputReader.next，并把对应类型的事件填入 mPendingEvent

gsensor

type 必须是 EV_ABS（绝对值），支持的 code 包括 EVENT_TYPE_ACCEL_X，EVENT_TYPE_ACCEL_Y EVENT_TYPE_ACCEL_Z，对应的 value 分别填 mPendingEvent 的 acceleration.x acceleration.y acceleration.z 三个成员 value 必须通过 CONVERT 常数来转换，这个值指是一个 G（重力加速度）相对的上报值

msensor

参考 data__poll_magnetic_sensor

type 必须是 EV_ABS（绝对值），支持的 code 包括

EVENT_TYPE_MAGV_X EVENT_TYPE_MAGV_Y EVENT_TYPE_MAGV_Z 对应的 value 填入 mPendingEvent 的 magnetic.x magnetic.y magnetic.z 成员 value 也要转换，转换的常数是一个特丝拉对应的上报值

osensor

同样参考 data__poll_magnetic_sensor（这个其实由 msensor 厂家提供）

type 必须是 EV_ABS（绝对值），支持的 code 包括 EVENT_TYPE_ORIENT_YAW EVENT_TYPE_ORIENT_PITCH EVENT_TYPE_ORIENT_ROLL 对应的 value 分别填入 mPendingEvent 的 orientation.azimuth orientation.pitch orientation.roll 三个成员 转换常数由 sansor 厂商定义，实际为 1 个弧度对应的值

psensor

参考 data__poll_proximity_sensor

type 同样是 EV_ABS（绝对值），支持 code EVENT_TYPE_PROXIMITY，value 为 1 或者 0，填入 mPendingEvent 的 distance 成员

lsensor

参考 data__poll_proximity_sensor

type 同样是 EV_ABS（绝对值），支持 code EVENT_TYPE_LIGHT，value 为上报光强度，填入 mPendingEvent 的 light 成员

4.4 加入 sensors_poll_context_t

sensorHAL 的起点在

struct sensors_module_t HAL_MODULE_INFO_SYM

有两个重要的实现

sensors__get_sensors_list 向 framework 提供支持的 sensor 的基本信息 第一步需要填写 sSensorList 这个数组

sensors_module_methods.open 成员 open_sensors 负责创建并初始化 sensors_poll_context_t 这个所有 sensor 的容器

在 sensors_poll_context_t 的构造函数中，我们可以看到创建各个 sensor 对象

```
mSensors[acc] = new Lis3dhSensor();
mSensors[akm] = new AkmSensor();
mSensors[al3006_pls] = new SensorAL3006();
```

在这里改成自己 sensor 的创建，同时，需要修改的还包括 handleToDriver 函数

```
case ID_A:
    return acc;
case ID_M:
case ID_O:
    return akm;
case ID_L:
case ID_P:
    return al3006_pls;
```

这几句建立了 framework 的 sensorID 到设备的对应关系

5. 结束语

请安装几个 sensor 的应用，通过 adb logcat 进行最后的调试，你会发现在解决几个粗心大意造成的小 bug 后，你的工作结束了

6. 篇外篇 1：G-sensor 的方向

G-sensor 的方向处理常常是个问题，各个厂商定义也不完全统一，其实调试阶段解决方法是统一的

adb shell 后 getevent 后观察上报的坐标

一般是

eventx 0003 0000 xxxx

eventx 0003 0001 xxxx

eventx 0003 0002 xxxx

这样一组，0003 是 EV_ABS 0000 0001 0002 分别是 x y z

好，首先平放，lcd 朝上，x y 都接近 0 z 是一个 g，如果 z 是负数，说明 gsensor 芯片是反贴的，请给 z 的数据转一下

其次，手机竖起来，这时候 z 和 x 应该是接近 0，y 应该是一个 g，注意如果 y 接近 0，而 x 值比较大，说明 gsensor 是 90 度贴的，请把 x y 的值交换

最后确认，手机竖时 y 位正，手机侧立右边朝下 x 为正，调一下 x y 的数据