密级状态：绝密(　　)　　秘密(　　)　　内部(　)　　公开(√)

Rockchip_Sensors 开发指南

（技术部，第二系统产品部）

文件状态： [　] 正在修改 [√] 正式发布	当前版本：	V1.0
	作　　者：	刘益星
	完成日期：	2018-6-5
	审　　核：	
	完成日期：	

福州瑞芯微电子股份有限公司

Fuzhou Rockchips Semiconductor Co., Ltd

版 本 历 史

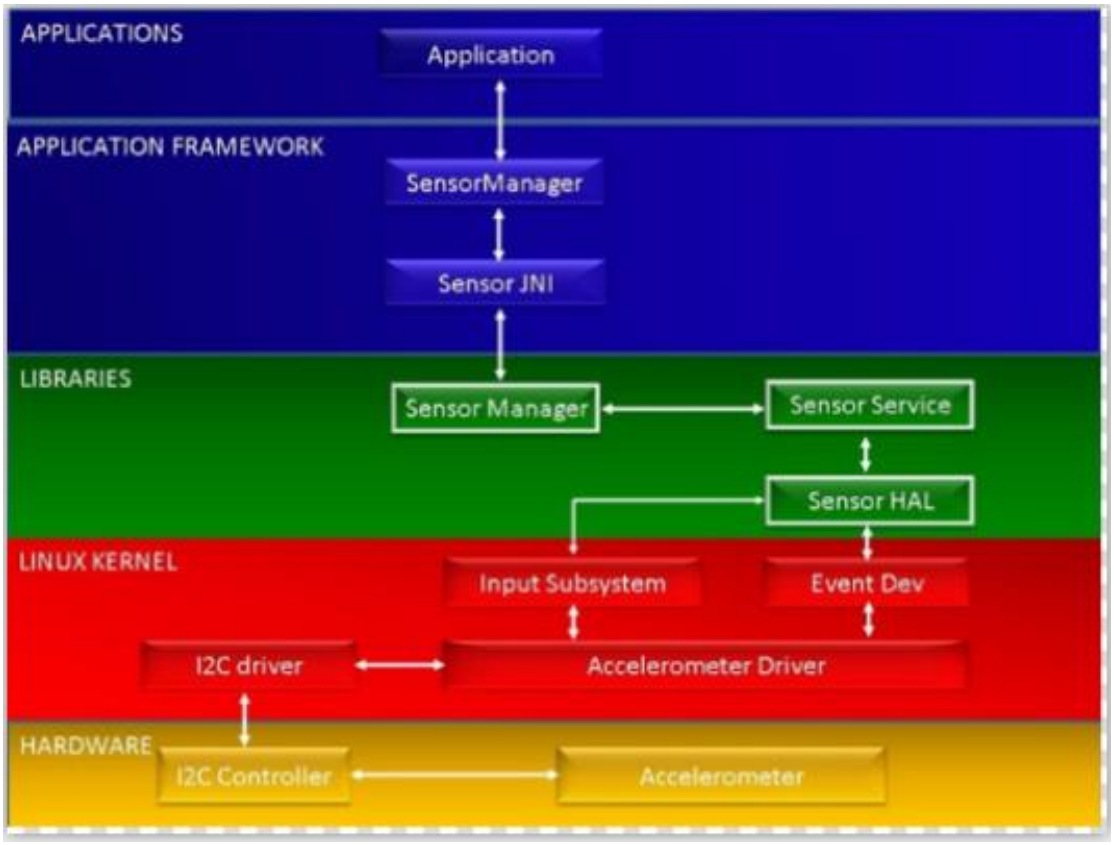
版本号	作者	修改日期	修改说明	备注
V1.0	刘益星	2018-6-5	发布初始版本	

目 录

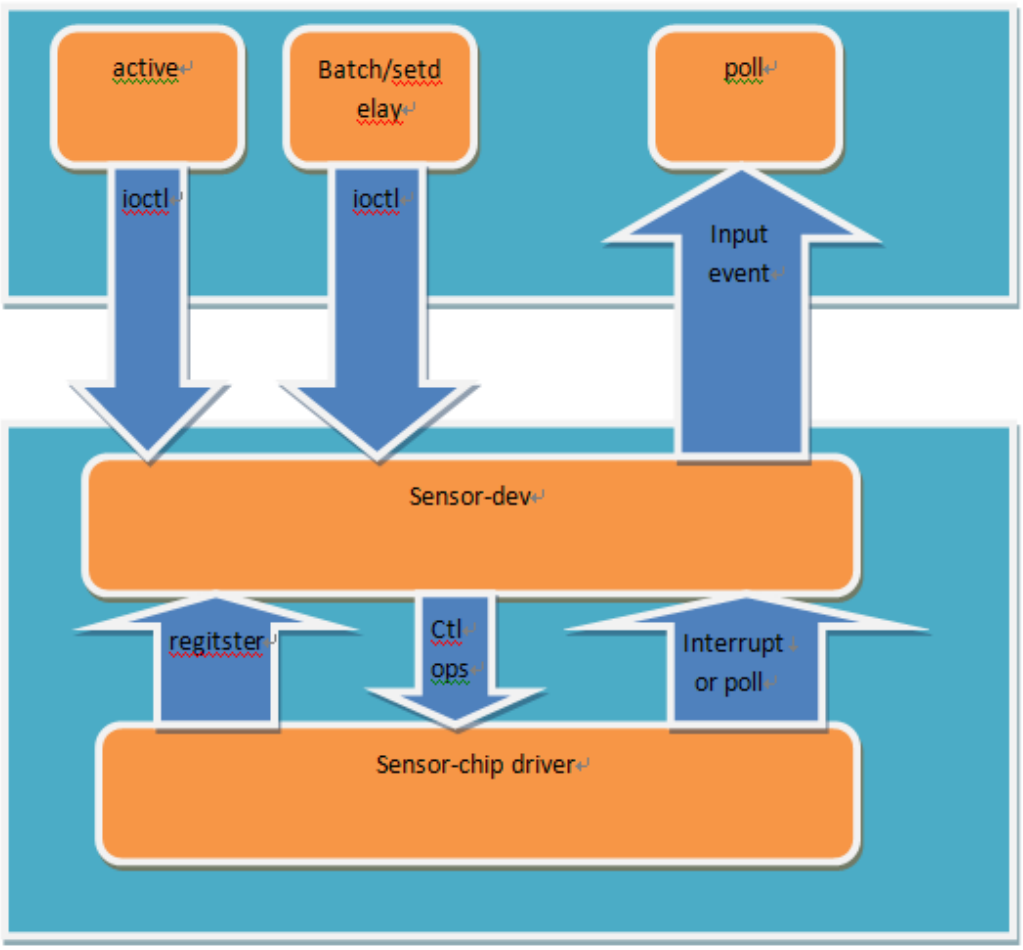
一、概述

本文主要介绍 rockchip sensors 的架构和基本配置；最新的 kernel4.4 内核对 sensors 的架构做了一些修改，以满足 android 的 cts 和 vts 测试要求；增加一个新的 sensor 驱动需做一些适配工作。

二、Android sensors 架构



三、Sensors hal 与 kernel driver 的通信框图



四、Rockchip sensors hal 介绍

代码路径：hardware/rockchip/sensor/st，支持最新的 SENSORS_DEVICE_API _VERSION_1_3 android sensors hal API 接口。其实现的接口包括：

get_sensors_list - 返回所有传感器的列表

activate - 启动或停止传感器

batch - 设置传感器的参数，如采样率和最大报告延迟

setDelay - 仅用于 1.0 版本的 HAL，设置指定传感器的采样率

flush - 刷写指定传感器的 FIFO ，并在完成后报告刷写完成事件

poll - 返回可用的传感器事件

其中 flush 接口并没有真正实现，因为要具体到硬件的 fifo 支持，由于我们支持的 sensor 型号多，无法做到统一接口实现，所以这里只是绕过，直接返回 META_DATA_FLUSH_COMPLETE 类型数据。

五、Rockchip sensors kernel driver 介绍

代码路径：kernel/drivers/input/sensors，其中 sensor-dev.c 是核心代码，整合了不同类型的 sensor，包括 accel, gyro, lsensor, psensor, compass 等。

六、新增一个 sensor 驱动流程

这里以增加一个最常用的 gsensor 为例，其他类型的 sensor 类似。

Android hal 不需要任何修改，hal 的是按+-2G 的 Gsensor 量程，adc 16bit 来配置的，也就是 1G 对应的数值是 16384；所以 driver 层增加 gsensor 驱动，上报数值需要根据量程和 adc 的精度来适配。

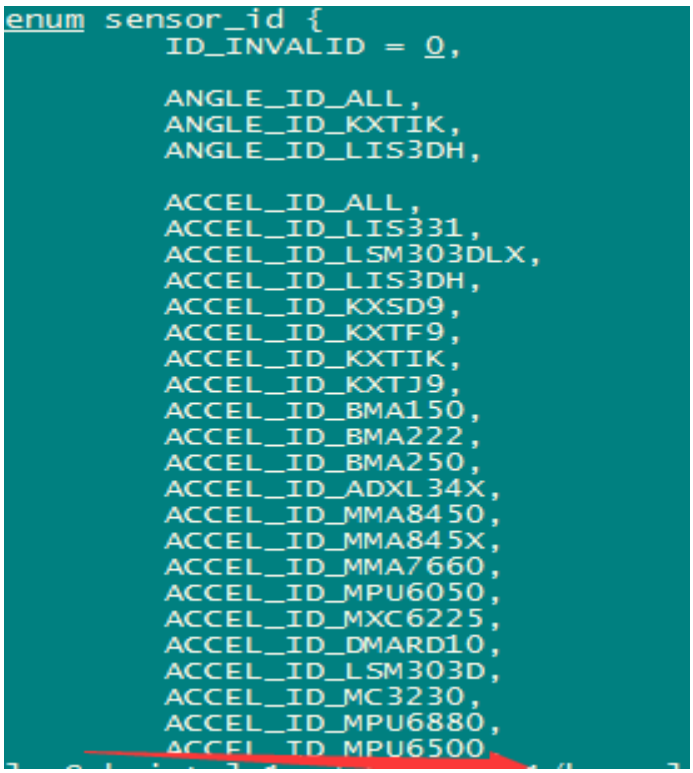
具体步骤如下，以 mpu6500 acc 为例：

6.1 增加 sensor id

sensor-dev.c 中添加 sensor_id，其中”mpu6500_acc”字段与 dts 中的 compatible 字段对应。

```
static const struct i2c_device_id sensor_id[] = {
        /*angle*/
        {"angle_kxtik", ANGLE_ID_KXTIK},
        {"angle_lis3dh", ANGLE_ID_LIS3DH},
        /*gsensor*/
        {"gsensor", ACCEL_ID_ALL},
        {"gs_mma8452", ACCEL_ID_MMA845X},
        {"gs_kxtik", ACCEL_ID_KXTIK},
        {"gs_kxtj9", ACCEL_ID_KXTJ9},
        {"gs_lis3dh", ACCEL_ID_LIS3DH},
        {"gs_mma7660", ACCEL_ID_MMA7660},
        {"gs_mxc6225", ACCEL_ID_MXC6225},
        {"gs_dmard10", ACCEL_ID_DMARD10},
        {"gs_lsm303d", ACCEL_ID_LSM303D},
        {"gs_mc3230", ACCEL_ID_MC3230},
        {"mpu6880_acc", ACCEL_ID_MPU6880},
        {"mpu6500_acc", ACCEL_ID_MPU6500},
        {"lsm330_acc", ACCEL_ID_LSM330},
        {"bma2xx_acc", ACCEL_ID_BMA2XX},
        {"gs_stk8baxx", ACCEL_ID_STK8BAXX},
        /*compass*/
        {"compass", COMPASS_ID_ALL},
        {"ak8975", COMPASS_ID_AK8975},
        {"ak8963", COMPASS_ID_AK8963},
        {"ak09911", COMPASS_ID_AK09911},
        {"mmc314x", COMPASS_ID_MMC314X},
```

ACCEL_ID_MPU6500 定义在 include/linux/sensor-dev.h 中：



6.2 增加 gsensor chip 驱动

driver/input/sensors/accel/目录下增加相应驱动

1）实现 gsensor_mpu6500_ops 实例：

struct sensor_operate gsensor_mpu6500_ops = {

.name = "mpu6500_acc", //与 sensor-dev.c 中定义的名称一致

.type = SENSOR_TYPE_ACCEL, //sensor 类型是 gsensor

.id_i2c = ACCEL_ID_MPU6500, // 定义在 include/linux/sensor-dev.h 中

.read_reg = MPU6500_ACCEL_XOUT_H, //读取 gsensor 数据的起始寄存器地址

.read_len = 6, //需要读取的 gsensor 数据的字节数

.id_reg = MPU6500_WHOAMI, //芯片唯一 ID 寄存器

.id_data = MPU6500_DEVICE_ID, //芯片 ID 值

.precision = MPU6500_PRECISION, //采样 gsensor 数据的

adc 位数

.ctrl_reg = MPU6500_PWR_MGMT_2, //用于使能 gsensor 的寄存器地址

.int_status_reg = MPU6500_INT_STATUS, //中断状态寄存器地址

.range = {-32768, 32768}, //量程，这里表示上报的量程为+-2G

.trig = IRQF_TRIGGER_HIGH |IRQF_ONESHOT, //中断类型

.active = sensor_active, //用于开关 gsensor

.init = sensor_init, //用于初始化 gsensor

.report = sensor_report_value, //用于上报 gsensor 数据

};

2）注册 sensor 驱动到 sensor-dev.c

```
static struct sensor_operate *gsensor_get_ops(void)
{
        return &gsensor_mpu6500_ops;
}

static int __init gsensor_mpu6500_init(void)
{
        struct sensor_operate *ops = gsensor_get_ops();
        int type = ops->type;

        return sensor_register_slave(type, NULL, NULL, gsensor_get_ops);
}

static void __exit gsensor_mpu6500_exit(void)
{
        struct sensor_operate *ops = gsensor_get_ops();
        int type = ops->type;
```

```
        sensor_unregister_slave(type, NULL, NULL, gsensor_get_ops);
}
```

内核驱动加载：

```
module_init(gsensor_mpu6500_init);
module_exit(gsensor_mpu6500_exit);
```

6.3 修改 gsensor range 和上报数据转化

不同厂家的 gsensor 在量程和精度上都会有差异，为了兼容不同厂家的 gsensor 芯片，同时做到不修改 android hal 层代码，只修改驱动，来达到简化客户开发的目的，我们必须把不同量程，不同精度的 gsensor 转换成统一标准的数据上报给 hal 层代码。

上节提到，HAL 层是以量程为+-2G，adc 16bit 的标准来上报的 gsensor 数据，以此标准不同的 gsensor 需要做适配。举例如下：

1）mpu6500 +- 2G 16bit adc

Mpu6500 的 adc 位数是 16bit 的，初始化的时候设置的量程是+-2G，那么寄存器读出来的数值范围为：+-32768，并且与上层 hal 层匹配，所以只要把寄存器读出来的值直接上报即可；

```
struct sensor_operate gsensor_mpu6500_ops = {
        .name                   = "mpu6500_acc",
        .type                   = SENSOR_TYPE_ACCEL,
        .id_i2c                 = ACCEL_ID_MPU6500,
        .read_reg               =
MPU6500_ACCEL_XOUT_H,
        .read_len               = 6,
        .id_reg                 = MPU6500_WHOAMI,
        .id_data                =
MPU6500_DEVICE_ID,
        .precision              =
```

```
MPU6500_PRECISION,
        .ctrl_reg                                   =
MPU6500_PWR_MGMT_2,
        .int_status_reg          = MPU6500_INT_STATUS,
        .range                   = {-32768, 32768},
        .trig                          = IRQF_TRIGGER_HIGH
|IRQF_ONESHOT,
        .active                  = sensor_active,
        .init                          = sensor_init,
        .report                         = sensor_report_value,
};
```

2）如果 gsensor 的量程和 adc 精度不是+-2G 和 16bit 的话，则需要做相应的转化，假设量程为+-XG，adc 是 Ybit 的，其转化关系如下：

```
#define XXX_RANGE        16384*X
#define XXX_PRECISION      Y
#define XXX_BOUNDARY       (0x1 << (XXX_PRECISION - 1))
#define XXX_GRAVITY_STEP   (XXX_RANGE / XXX_BOUNDARY)

struct sensor_operate gsensor_xxx_ops = {
        .name                    = "xxx_acc",
        .type                    = SENSOR_TYPE_ACCEL,
        .id_i2c                  = ACCEL_ID_XXX,
        .read_reg                       = XXX_ACCEL_XOUT_H,
        .read_len                       = 6,
        .id_reg                  = XXX_WHOAMI,
        .id_data                        = XXX_DEVICE_ID,
        .precision                      = Y,
        .ctrl_reg                       = XXX_PWR_MGMT_2,
        .int_status_reg          = XXX_INT_STATUS,
        .range                   = {-16384*X, +16384*X},
```

```
        .trig                                   = (IRQF_TRIGGER_LOW |
IRQF_ONESHOT),
        .active                         = sensor_active,
        .init                                   = sensor_init,
        .report                         = sensor_report_value,
};
```

上报值转化：

report_data = read_data* XXX_GRAVITY_STEP; //read_data 为实际寄存器读出来的值，report_data 为转换后可以上报给 hal 层的值

6.4 修改 gsensor layout 方向

Android 系统对坐标轴的定义如下：

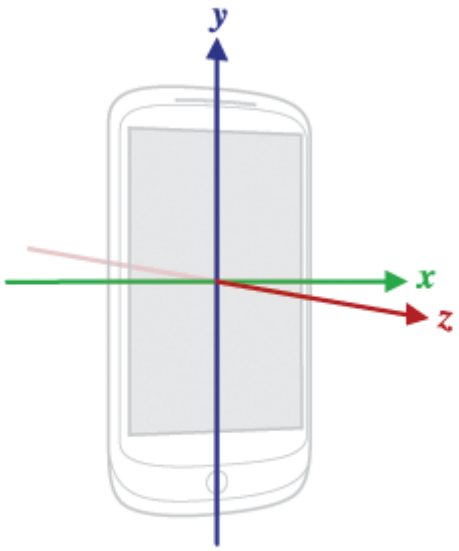


图 1. Sensor API 使用的坐标系（相对于移动设备）。

汽车坐标轴

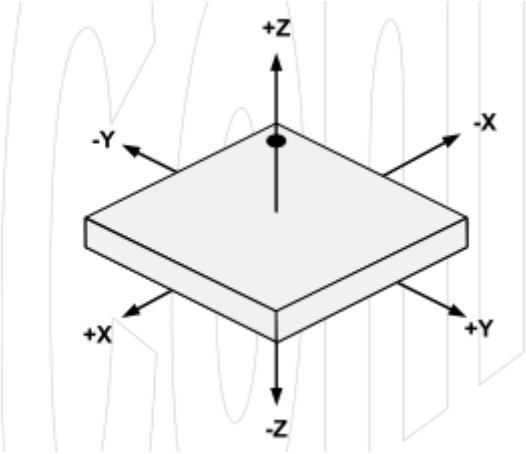
在 Android Automotive 实现中，坐标轴相对于车身坐标系进行定义：

图 2. Sensor API 使用的坐标系（相对于汽车设备）。

- X 轴沿车辆右侧延伸
- Y 轴沿车架前方延伸
- Z 轴沿车架顶部延伸

从坐标轴的正方向观察时，正旋转方向为逆时针方向。因此，当车辆向左转时，z 轴的陀螺仪转速应该为正值。

Sensor chip 的坐标系定义：



每一颗 sensor 都有自己的坐标系定义，所以需要根据 layout 方向，把 sensor 的坐标系转换成 android 设备的坐标系，这个可以通过一个矩阵来做转换；

如下矩阵表示：

$$\begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & -1 \end{pmatrix}$$

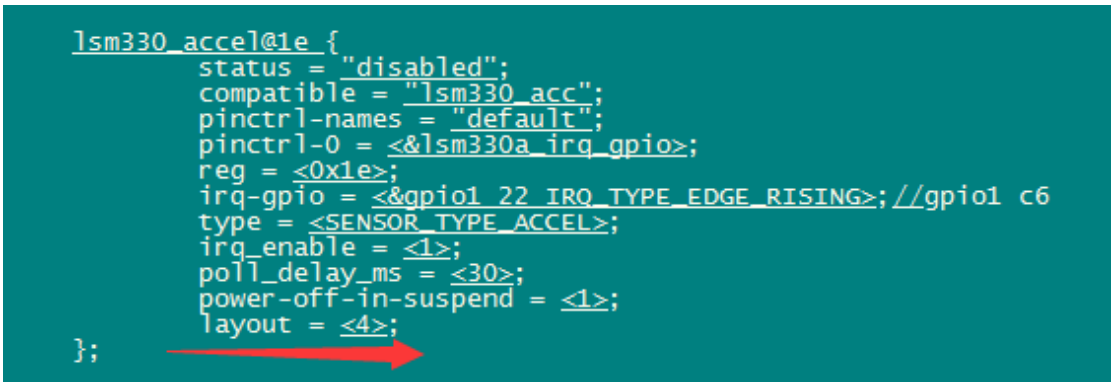
X = x*0 + y*1 + z*0 = y

Y = x*1 + y*0 + z*0 = x

Z = x*0 + y*0 + z*-1 = -z

Sensor-dev.c 定义了 layout 值，对应不同的矩阵，具体可以查看 sensor-dev.c 代码查询，所以，适配方向可以通过修改 dts 里面的 layout 值来选择对应的转换矩阵，达到方向的变化，以适配 android 的要求。

如下：



6.5 dts 中其他值的意义

lsm330_accel@1e {

status = "disabled";

compatible = "lsm330_acc"; //与 sensor-dev.c 中的 sensor_id 定义匹配

pinctrl-names = "default"; //相关 pinctrl 定义，这里主要定义中断脚

pinctrl-0 = <&lsm330a_irq_gpio>;

reg = <0x1e>; //i2c 地址

irq-gpio = <&gpio1 22 IRQ_TYPE_EDGE_RISING>;//中断脚，中断类型

type = <SENSOR_TYPE_ACCEL>; //传感器类型，不能搞错

irq_enable = <1>; //是否使用中断模式，如果要过 cts 或者 vts 建议使用轮询，sensor-dev.c 的轮询模式已经可以满足 cts 测试要求，如果使用中断模式，需要 gsensor chip driver 做好采样率的配置；

poll_delay_ms = <30>; //轮询间隔，最新代码这个值没有意义

了，代码里面会根据上层的配置修改这个值

```
                layout = <4>;
                    reprobe_en = <1>;  //如果这个为 1，表示在 sensor 驱动加载失败后，会多次尝试重新 sensor 驱动（次数限制定义在代码里）
        };
```

6.6 Android 中的 sensor 相关宏配置

BoardConfig.mk 中：

```
# Sensors
BOARD_SENSOR_ST := true    //采用 RK 的 sensors Hal，也就是本文介绍的
BOARD_SENSOR_MPU_PAD := false   //仅适用 MPU6500、mpu6050 等芯片
```

支持哪些类型的 sensor，如果没有，要配置成 false，否则 vts 和 cts 测试会失败：

```
BOARD_GRAVITY_SENSOR_SUPPORT := true
BOARD_COMPASS_SENSOR_SUPPORT := false
BOARD_GYROSCOPE_SENSOR_SUPPORT := false
BOARD_PROXIMITY_SENSOR_SUPPORT := false
BOARD_LIGHT_SENSOR_SUPPORT := true
BOARD_PRESSURE_SENSOR_SUPPORT := false
BOARD_TEMPERATURE_SENSOR_SUPPORT := false
```

七、Gsensor 和 gyro 的校准

Gsensor 和 gyro 的校准可通过命令行方式，pcba 测试的时候也可以做校准，具体查看 pcba 的配置。

命令行校准方法：保持机器水平静止放置，输入以下命令校准：

```
Gsensor: echo 1 > /sys/class/sensor_class/accel_calibration
GYRO：echo 1 > /sys/class/sensor_class/gyro_calibration
```

查看校准值：

```
cat /sys/class/sensor_class/accel_calibration
```

cat /sys/class/sensor_class/gyro_calibration

如果无法查看校准值，则说明校准失败，可以打印 kernel log 确定失败原因。

校准成功后，校准的值会保存到 nand 或 emmc 的 vendor storage 里面，不会被擦除，开机自动生效。

八、Sensor 常见问题分析

这里以 gsensor 为例进行介绍，其他类型 sensor 可参考这里。

8.1 Gsensor 功能无法使用

8.1.1 getevent 查看 gsensor 无数据上报

排查思路：

1）确认驱动是否注册上，可以通过 getevent 查看是否有名为 gsensor 的 input 设备；没有的话就要看为何没注册上，有可能是 dts 配置问题，驱动问题，i2c 同时失败，chip id 不匹配等等可能的原因，查看 kernel log，具体问题具体分析；如果有注册上，往下。

2）确认 android 层相关宏是否打开；

3）确认 hal 层代码是否正常运行；开机打印 logcat，搜索 sensor 字段查看相关 log。

8.1.2 数据值不对

Getevent 有数据上报，屏幕不旋转或者旋转无规律；

除可通过 sensor 的 apk 查看数值是否正确外，还可以通过命令的方式查看 android 上报的 gsensor 数据是否正确：

setprop sensor.debug.level 2

手动灭屏休眠，再唤醒后命令生效，通过 logcat –s SensorsHal 可以查看到 hal 上报的 gsensor 数值；

静止水平放置时，理论正确的数值应该是有个轴的值为+9.8 或者-9.8，另外两个轴的值为 0；

如果数值不对，请确认上面章节介绍的数据上报转换是否正确。

8.2 方向不对

方向不对的话，通过修改 dts 的 layout 值，参考上面的 6.4 章节。

8.3 cts/vts 测试不过

1）数值偏差大，可以通过 gsensor 校准后再做测试；

类似此种错误：

arm64-v8a VtsHalSensorsV1_0Target

Test	Result	Details
HidlHalGTest#SensorsHidlTest.AccelerometerStreamingOperationSlow_64bit	fail	hardware/interfaces/sensors/1.0/vts/functional/VtsHalSensorsV1_0TargetTest.cpp:966 Value of: checker.check(events, &s) Actual: false Expected: true Event @ 1703624402490 (0, 0.0155623, 0) has norm 0.0155623, which is beyond range [9.30665, 10.3067]

Gsensor 校准方法：机器水平静止放置，通过命令 echo 1 > /sys/class/sensor_class/accel_calibration，查看校准是否成功，cat /sys/class/sensor_class/accel_calibration，查看是否有值，读取失败则说明校准失败，可以查看 kernel log 确认失败原因。

如果 gsensor 偏差太大，是无法校准过的，此时可以适当放宽校准通过的判定值再次校准看是否能过，判定值修改方法，修改代码 kernel/drivers/input/sensor/sensor-dev.c 里面的 ACCEL_OFFSET_MAX 值：

#define ACCEL_OFFSET_MAX 1600　//最大建议不超过 2500

这个值也不能放的太大，否则就失去校准的意义了。如果是 sensor 芯片异常，或者硬件干扰太大，还是要从硬件上去解决。

2）采样率不够

类似下面的错误就是采样率不够：

armeabi-v7a CtsSensorTestCases

Test	Result	Details
android.hardware.cts.SensorBatchingTests#testAccelerometer_50hz_batching	fail	junit.framework.AssertionFailedError: VerifySensorOperation | sensor='Accelerometer sensor', samplingPeriod=20000us, maxReportLatency=10000000us | 53 events gaps: position=12, delta_time=46.72ms; position=26, delta_time=46.83ms; position=42, delta_time=52.16ms; 50 more; (expected <36.00ms), Frequency out of range: Requested "Accelerometer sensor" at 50.00hz (expecting between 45.00Hz and 110.00Hz, measured 38.04Hz)
android.hardware.cts.SensorBatchingTests#testAccelerometer_50hz_flush	fail	junit.framework.AssertionFailedError: VerifySensorOperation | sensor='Accelerometer sensor', samplingPeriod=20000us, maxReportLatency=10000000us | Frequency out of range: Requested "Accelerometer sensor" at 50.00hz (expecting between 45.00Hz and 110.00Hz, measured 37.95Hz), 17 events gaps: position=4, delta_time=46.58ms; position=5, delta_time=46.62ms; position=19, delta_time=116.76ms; 14 more; (expected <36.00ms)
android.hardware.cts.SensorBatchingTests#testAccelerometer_fastest_batching	fail	junit.framework.AssertionFailedError: VerifySensorOperation | sensor='Accelerometer sensor', samplingPeriod=0us, maxReportLatency=10000000us | Frequency out of range: Requested "Accelerometer sensor" at fastest (expecting between 128.57Hz and 314.29Hz, measured 85.83Hz), Failed due to (insufficient events 1115/1257,), 153 events gaps: position=9, delta_time=20.48ms; position=10, delta_time=19.57ms; position=11, delta_time=20.10ms; 150 more; (expected <12.60ms)
android.hardware.cts.SensorBatchingTests#testAccelerometer_fastest_flush	fail	junit.framework.AssertionFailedError: VerifySensorOperation | sensor='Accelerometer sensor', samplingPeriod=0us, maxReportLatency=10000000us | Frequency out of range: Requested "Accelerometer sensor" at fastest (expecting between 128.57Hz and 314.29Hz, measured 88.18Hz), 55 events gaps: position=5, delta_time=19.92ms; position=12, delta_time=19.85ms; position=14, delta_time=20.15ms; 52 more; (expected <12.60ms)
android.hardware.cts.SensorTest#testSensorTimeStamps	fail	junit.framework.AssertionFailedError: VerifySensorOperation | sensor='Accelerometer sensor', samplingPeriod=0us, maxReportLatency=0us | 227 events gaps: position=7, delta_time=13.32ms; position=15, delta_time=13.22ms; position=16, delta_time=13.31ms; 224 more; (expected <12.60ms)
android.hardware.cts.SingleSensorTests#testAccelerometer_100hz	fail	junit.framework.AssertionFailedError: VerifySensorOperation | sensor='Accelerometer sensor', samplingPeriod=10000us, maxReportLatency=0us | 40 events gaps: position=5, delta_time=26.61ms; position=8, delta_time=53.45ms; position=14, delta_time=26.72ms; 37 more; (expected <18.00ms), Frequency out of range: Requested "Accelerometer sensor" at 100.00hz (expecting between 90.00Hz and 220.00Hz, measured 66.62Hz)
android.hardware.cts.SingleSensorTests#testAccelerometer_10hz	fail	junit.framework.AssertionFailedError: VerifySensorOperation | sensor='Accelerometer sensor', samplingPeriod=100000us, maxReportLatency=0us | 3 events gaps: position=3, delta_time=309.94ms; position=4, delta_time=309.82ms; position=14, delta_time=206.60ms; (expected <180.00ms), Frequency out of range: Requested "Accelerometer sensor" at 10.00hz (expecting between 9.00Hz and 22.00Hz, measured 8.24Hz)
android.hardware.cts.SingleSensorTests#testAccelerometer_15hz	fail	junit.framework.AssertionFailedError: VerifySensorOperation | sensor='Accelerometer sensor', samplingPeriod=66667us, maxReportLatency=0us | 6 events gaps: position=3, delta_time=139.81ms; position=40, delta_time=140.37ms; position=47, delta_time=139.95ms; 3 more; (expected <120.00ms), Frequency out of range:

目前 RK 的 sensor 驱动，采用轮询的方法，基本可以满足 android 采样率的要求，所以如果不知道如何动态配置采样率的话，建议采用轮询方法，同时芯片初始化的时候就把采样率配置成高采样率，200Hz 就可以；另外，Sensor 芯片驱动中最好不要对数据做过滤，如果有，可以去掉数据过滤直接上报再测试看看；还有一种情况是芯片本身的采样率实在上不去，比较 low 的芯片；只能通过软件来绕，当 android 要求的采样率大于 sensor 的采样率，意味着存在多次读取的数值一样的情况，同样数值在通过 input 上报的时候，会被框架层丢弃，这样就达不到 android 的测试要求，此时我们可以人为修改数值，通过软件骗过框架，如下：

```
/*
 *input dev will ignore report data if data value is the same with last_value,
 *sample rate will not enough by this way, so just avoid this case
 */
if ((sensor->axis.x == axis.x) && (sensor->axis.y == axis.y) && (sensor->axis.z == axis.z)) {
        if (flag) {
                flag = 0;
                axis.x += 1;
                axis.y += 1;
                axis.z += 1;
        } else {
                flag = 1;
                axis.x -= 1;
                axis.y -= 1;
                axis.z -= 1;
        }
}
```

具体可以查看 drivers/input/sensor/accel/mma7660.c 中的 sensor_report_value 函数。