Android Input Sensor
开发指南

版本号: 2.1
发布日期: 2025.01.20

版本历史

版本号	日期	制/修订人	内容描述
1.0	2020.11.11	AW	初始版本文档
1.1	2020.12.31	AW	更新部分配置路径
2.0	2023.03.16	AW1696	1. 增加 sensorhal 介绍。2. 完成 sensor 适配过程。3. 增加调试及常见问题。
2.1	2025.01.20	AW2078	title 增加 Input 字段。

目　录

1 前言

1.1 编写目的

为达到能快速使用的目的。文档主要对 Input Sensor 的使用方法步骤，如何在 SensorHal 添加一个新的模组等做了详细的讲解。

1.2 适用范围

介绍本模块设计适用 AW 平台。

1.3 相关人员

相关的开发与维护人员应该仔细阅读本文档。

2 驱动模块适配

驱动模块的适配，参考《Android Input 开发指南》；本文档主要介绍 Sensorhal 的开发适配。

3 Sensorhal 介绍

SensorHal 是 google 原生支持的 Hardware 之一，主要用于管理 Linux Sensor，获取 event 并传递到 Framework，以供应用层使用，例如实现自动旋转，光距感应等。

在 AW 发布的 SDK 中，Android 13 以上 Sensorhal 使用 AIDL，在以往的版本使用 HIDL，本文主要基于 AIDL 进行介绍，对于与 HIDL 不一致且需要开发注意的不同点，会特别注明。

3.1 框架介绍

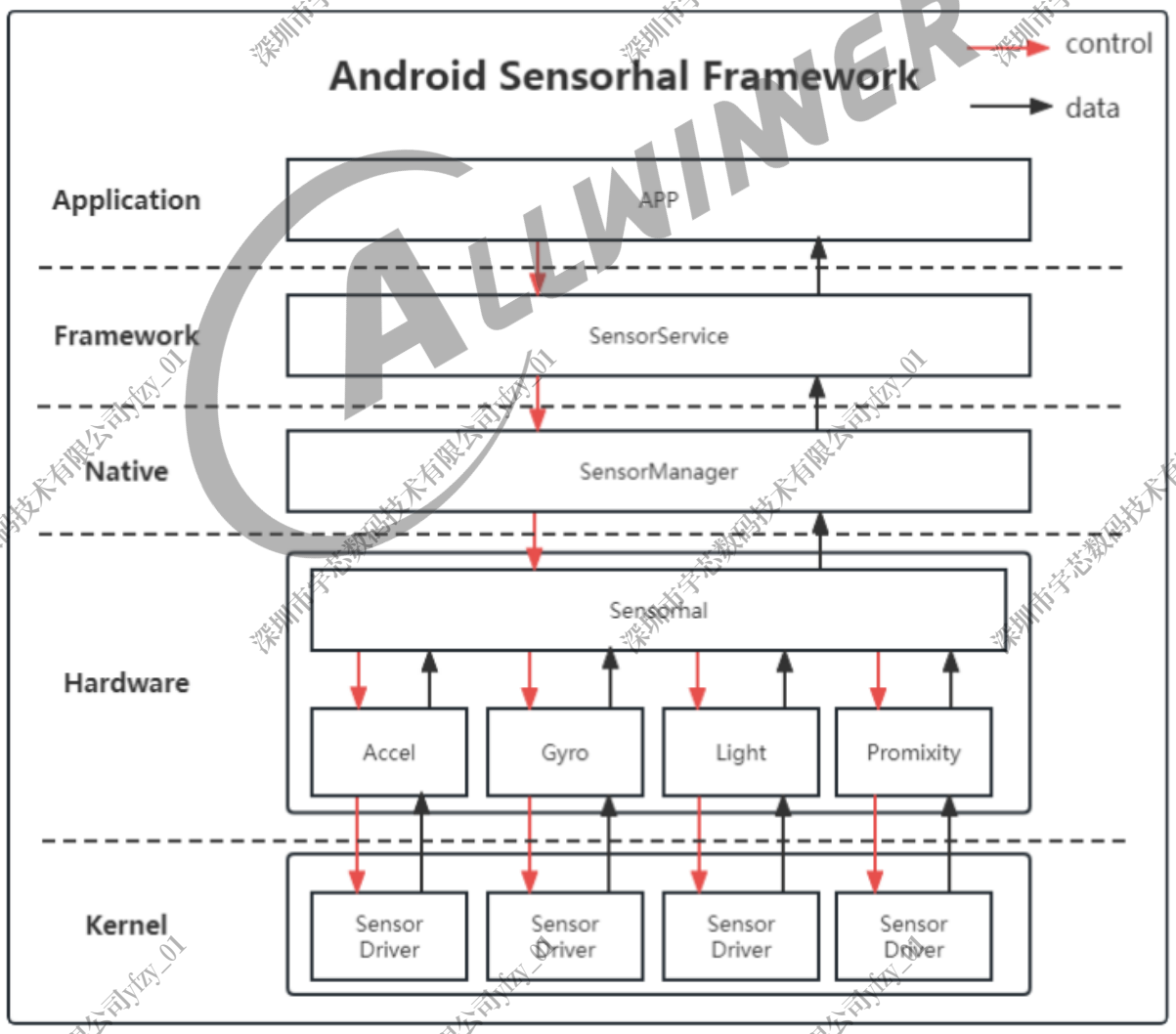

图 3-1: Sensor 框架

上图是 Android 中 Sensor 的控制-数据流程图，可看到 Sensorhal 的位置及作用，是处于一个承上启下的作用。

3.2 源码介绍

3.2.1 Google SensorHal AIDL

Google 原生代码路径，通常有 default 实现，支持的 Sensor 类型，Sensor 数据头文件等。

```
/hardware/interfaces/sensors/
```

aidl 源文件路径。

```
hardware/interfaces/sensors/aidl/android/hardware/sensors
```

其文件描述信息如下。

```
└── sensors
    ├── AdditionalInfo.aidl
    ├── DynamicSensorInfo.aidl
    ├── Event.aidl                  //event事件定义
    ├── ISensors.aidl               //SensorHal的主文件，定义了Sensorhal的使用接口
    ├── ISensorsCallback.aidl
    ├── SensorInfo.aidl             //SensorInfo数据介绍
    ├── SensorStatus.aidl           //Sensor状态描述
    └── SensorType.aidl             //Sensor类型介绍，定义了当前支持的所有Sensor类型
```

3.2.2 AW Sensorhal

全志当前 Sensorhal 源码路径，主要是 Sensorhal 的实现。

```
hardware/aw/sensors/aw_sensors
```

其源码实现。

```
aidl/
├── impl                            //Sensorhal实现
│   ├── AccelSensor.cpp             //重力感应gsensor实现类
│   ├── AmbientTempSensor.cpp
│   ├── Android.bp
│   ├── include                     //头文件
│   ├── InputEventReader.cpp        //用于读取Event事件
│   ├── LightSensor.cpp
│   ├── MagnetometerSensor.cpp
│   ├── PressureSensor.cpp
│   ├── ProximitySensor.cpp
│   ├── Sensor.cpp                  //Sensor的父类，定义了一个Sensor的标准接口
│   ├── SensorInit.cpp              //Sensor初始化，保存了预定义Sensor信息，具有加载Sensor驱动，初始化Sensor信息
```

```
    功能
    │   └── Sensors.cpp            //Sensorhal的主要接口，用于管理所有的Sensor
    └── service                     //service实现
        ├── Android.bp
        ├── main.cpp
        ├── sensors-default-aw.rc   //sensor启动rc类
        └── sensors-default-aw.xml  //兼容性矩阵
```

3.2.3 sensors.h 介绍

senosrs.h 是 google 原生的 sensor 库文件，主要是定义了许多 sensor 相关的信息，路径如下。

```
/hardware/libhardware/include/hardware/sensors.h
```

这里介绍常用的一些信息。

3.2.3.1 SENSOR_STRING_TYPE

定义了 sensor 的 string 名称，用于 SensorInfo 上报。

```
#define SENSOR_STRING_TYPE_ACCELEROMETER      "android.sensor.accelerometer"
#define SENSOR_STRING_TYPE_MAGNETIC_FIELD     "android.sensor.magnetic_field"
#define SENSOR_STRING_TYPE_ORIENTATION      "android.sensor.orientation"
#define SENSOR_STRING_TYPE_GYROSCOPE      "android.sensor.gyroscope"
#define SENSOR_STRING_TYPE_LIGHT      "android.sensor.light"
#define SENSOR_STRING_TYPE_PRESSURE      "android.sensor.pressure"
#define SENSOR_STRING_TYPE_TEMPERATURE      "android.sensor.temperature"
#define SENSOR_STRING_TYPE_PROXIMITY      "android.sensor.proximity"
```

3.2.3.2 sensor_t

该结构体主要用于保存 sensor 信息。

```
struct  sensor_t {
    const char*     name;               //sensor名字
    const char*     vendor;             //产品，一般描述是哪家的
    int             version;            //版本号
    int             handle;             //用于索引
    int             type;               //sensor类型
    float           maxRange;           //最大范围
    float           resolution;         //分辨率
    float           power;              //通常为0，当前未使用
    int32_t         minDelay;           //最小采样间隔时间，单位为ns
    uint32_t        fifoReservedEventCount;
    uint32_t        fifoMaxEventCount;
    const char*     stringType;         //sensor类型的string，参考<SENSOR_STRING_TYPE>章节
    const char*     requiredPermission; //需要的权限，默认为0

    #ifdef __LP64__
```

```
    int64_t maxDelay;                   //最大采样间隔时间，单位为ns
#else
    int32_t maxDelay;
#endif

#ifdef __LP64__
    uint64_t flags;                     //flag，通常有wakeup,contious,on-change等
#else
    uint32_t flags;
#endif

    void*           reserved[2];
};
```

3.3 Sensor 类型

当前 Android 版本支持的主要常用 Sensor 类型如下。

```
ACCELEROMETER;          //重力感应gsensor
MAGNETIC_FIELD;         //磁力感应
GYROSCOPE;              //陀螺仪
LIGHT;                  //光感
PROXIMITY;              //距离感应
```

支持的类型 type 描述信息具体可查阅如下文件。

```
hardware/interfaces/sensors/aidl/android/hardware/sensors/SensorType.aidl
```

说明

霍尔传感器不属于常规的 Sensor，霍尔传感器适配请参考《Android Input 开发指南》。

3.4 SensorInfo

在 Sensorhal 中，需要预先定义使用的 Sensor 信息用于初始化，当前所有可能使用的 Sensor 信息存放位置如下。

```
hardware/aw/sensors/aw_sensors/aidl/impl/SensorInit.cpp
```

Sensor 信息是以结构体数组存放的，参考定义如下。

```
struct SensorData AccelSensorConfigList[] = {
}
```

其中，SensorData 的定义如下。

```
struct SensorData {
    char sensorName[64];            //Sensor的名字，表现为dev/input/下的名字
    char classPath[128];            //Sensor的sys/class/input/inputX路径，这里均为空，运行时获取
    char enableNodeName[64];        //Sensor的enable节点名字，使能时与classPath拼接使用
```

```
    char delayNodeName[64];         //Sensor的delay节点名字，使能时与classPath拼接使用
    float  priData;                 //Sensor的私有数据，用于sensor的数组转换，如gsensor的转换数据等
    struct  sensor_t sensorInfo;    //Sensor信息
};
```

在 SensorData 中还存在 sensor_t 结构体，可参考章节。

如下是一个 AccelSensor 举例。

```
    {
        // sc7a20
        "sc7a20",                           //dev/input下的驱动名称
        "",                                 //sys/class/input下的路径，默认空，自动生成
        DEFAULT_ENABLE,                     //enable节点，默认为"enable"
        DEFAULT_DELAY,                      //delay节点，默认为"delay"
        32.0f,                              //pri数据，用于转换上报的事件
        {                                   //sensor_t结构体
            "lis7a20 detect Accelerometer", //产品名称
            "SL",                           //产品供应商，这里是SL
            1,                              //版本为1
            0,                              //handle，填0，自动生成
            SENSOR_TYPE_ACCELEROMETER,      //sensor类型
            (GRAVITY_EARTH * 4.0f),         //最大范围为4倍重力
            GRAVITY_EARTH / 1024.0f,        //最小分辨率
            0.2f,                           //power，当前未使用
            10000,                          //最小采样间隔，单位ns，这里则是10ms
            0,                              //fifoReservedEventCount,为0即可
            0,                              //fifoMaxEventCount,为0即可
            SENSOR_STRING_TYPE_ACCELEROMETER,//参考<SENSOR_STRING_TYPE>
            0,                              //权限，为0即可
            1000000,                        //最大采样间隔，这里是1s
            SENSOR_FLAG_CONTINUOUS_MODE,    //flags，这里为continous
            {},                             //reserved，空即可
        },
    },  // sc7a20
```

3.5 SensorHal 工作流程

3.5.1 初始化

初始化工作流程如图所示。

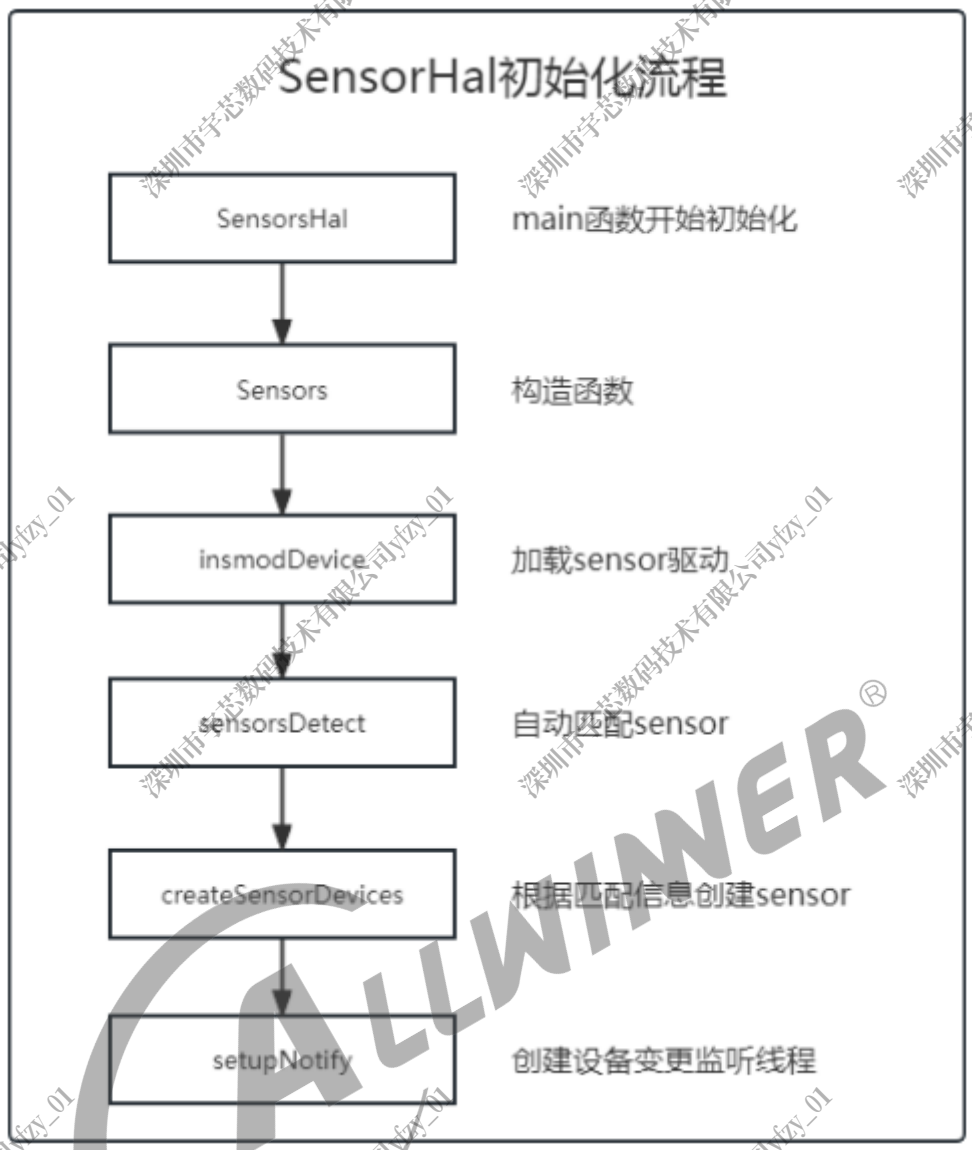

图 3-2: Sensor 框架

3.5.1.1 Sensor 驱动加载

Android 系统启动过程中，会启动 hal，Sensorhal 也会在此阶段被启动。

Sensorhal 启动过程，会加载sensor_modules.cfg中的驱动。

其驱动加载函数为insmodDevice

sensor_modules.cfg在方案目录中进行配置。

```
device/softwinner/xxx/xxx/input/sensor_modules.cfg
```

使用如下。

```
preinsmod=init-input.ko

handle=ID_A
path=sc7a20.ko

handle=ID_A
path=mir3da.ko

path=mmc5603.ko
```

- preinsmod 为依赖模块，即 sensor 模块加载需要先加载的驱动。
- handle 为预定义的 handle 信息，当前 AIDL 版本已废弃。
- path 为模块名字，会自动寻找/vendor/lib/modules/下的对应驱动。

说明

其中，handle 在 AIDL 版本中已废弃，在 HIDL 2.0 版本中还有使用，定义在hardware/aw/sensors/aw_sensors/2.0/hal/include/sensors.h可自行查阅。

3.5.1.2 自动匹配设备

加载完驱动后，sensorhal 初始化过程会遍历 sys/class/input 目录，根据设备名自动匹配预定义的 sensor 信息，如匹配成功，则保存对应信息，用于后续 sensorhal 初始化。其函数为sensorsDetect。

预定义的 sensor 信息定义在如下路径，其详细定义参考《SensorInfo》章节。

```
hardware/aw/sensors/aw_sensors/aidl/impl/SensorInit.cpp
```

3.5.1.3 创建 sensor

获取到匹配信息后，即可自动创建设备，实现如下。

```
void Sensors::createSensorDevices() {
    for (int32_t i = 0; i < gSensorCount; i++) {
        switch (gSensorData[i].sensorInfo.type) {
            case SENSOR_TYPE_ACCELEROMETER:
                AddSensor<AccelSensor>(gSensorData[i], mNextHandle++);
                AddSensor<AccelUncalSensor>(gSensorData[i], mNextHandle++);
                break;

            case SENSOR_TYPE_LIGHT:
                AddSensor<LightSensor>(gSensorData[i], mNextHandle++);
                break;

            case SENSOR_TYPE_PROXIMITY:
                AddSensor<ProximitySensor>(gSensorData[i], mNextHandle++);
                break;

            case SENSOR_TYPE_MAGNETIC_FIELD:
                AddSensor<MagnetometerSensor>(gSensorData[i], mNextHandle++);
```

```
                AddSensor<MagnetometerUncalSensor>(gSensorData[i], mNextHandle++);
                break;

            case SENSOR_TYPE_PRESSURE:
                AddSensor<PressureSensor>(gSensorData[i], mNextHandle++);
                break;

            case SENSOR_TYPE_AMBIENT_TEMPERATURE:
                AddSensor<AmbientTempSensor>(gSensorData[i], mNextHandle++);
                break;

            default:
                ALOGD("SensorType %d is not support!\n", gSensorData[i].sensorInfo.type);
                break;
        }
    }
}
```

部分 sensor 未进行适配，可参考上述函数进行即可。

说明

对于部分 Sensor，谷歌 GMS 要求实现 Uncal（未校准）类型，例如 AccelUncalSensor，其实现也是在 AccelSensor.cpp 里。

3.5.2 使能 Sensor

使能 sensor 是通过 Sensor 类中的 activate 方法。

```
void Sensor::activate(bool enable) {
    if (mIsEnabled != enable) {
        mIsEnabled = enable;
        setEnable(enable);
        mWaitCV.notify_all();
    }
}

void Sensor::setEnable(bool enable) {
    char buf[2];

    ALOGD("%s set enable = %d\n", mSensorName, enable);
    if (mClassPath == "" || mEnableNodeName == "") {
        ALOGE("%s classPath %s enable node %s error, please check!",
                mSensorName, mClassPath, mEnableNodeName);
    }
    int len = sprintf(buf, "%d", enable);

    setSysfsInputAttr(mClassPath, mEnableNodeName, buf, len);
    return;
}
```

activate 则会调用 setEnable 方法，setEnable 通过自动检索的 input 路径及预定义好的 enable 使能节点，自动拼接/sys/class/input/inputX/enable，并写入对应的值。

说明

有特殊需求的，可重新 setEnable 方法。

3.5.3 设置 delay

设置 delay 则是通过 batch 方法。

```
void Sensor::batch(int64_t samplingPeriodNs) {
    if (samplingPeriodNs < mSensorInfo.minDelayUs * 1000LL) {
        samplingPeriodNs = mSensorInfo.minDelayUs * 1000LL;
    } else if (samplingPeriodNs > mSensorInfo.maxDelayUs * 1000LL) {
        samplingPeriodNs = mSensorInfo.maxDelayUs * 1000LL;
    }

    mSamplingPeriodNs = samplingPeriodNs;
    // set delay
    setSamplingInterval(samplingPeriodNs);
    // Wake up the 'run' thread to check if a new event should be generated now
    mWaitCV.notify_all();
}

void Sensor::setSamplingInterval(int64_t samplingPeriodNs) {
    char buf[80];
    int len;

    ALOGD("%s set samplingPeriodNs = %" SCNi64, mSensorName, samplingPeriodNs);
    if (mClassPath == "" || mDelayNodeName == "") {
        ALOGE("%s classPath %s delay node %s error, please check!",
              mSensorName, mClassPath, mDelayNodeName);
        return;
    }
    len = sprintf(buf, "%" SCNi64, samplingPeriodNs / 1000000); /*convert ns to ms*/

    setSysfsInputAttr(mClassPath, mDelayNodeName, buf, len);
}
```

batch 会自动判断设置的 delay 是否在允许的范围内，允许则会调用 setSamplingInterval 完成 delay 的设置。setSamplingInterval 通过自动检索的 input 路径及预定义好的 delay 节点，自动拼接/sys/class/input/inputX/delay，并写入对应的值。

说明

有特殊需求的，可重新 setSamplingInterval 方法。

3.5.4 处理 event

读取 event 是通过 readEvents 方法完成。

```
std:: vector<Event> Sensor::readEvents() {
    std:: vector<Event> events;
    Event event;
    event.sensorHandle = mSensorInfo.sensorHandle;
    event.sensorType = mSensorInfo.type;
    memset(&event.payload, 0, sizeof(event.payload));
    readEventPayload(event.payload);
    event.timestamp = ::android::elapsedRealtimeNano();
    events.push_back(event);
```

```
    return events;
}
```

而 sensor 中要实现的则是 readEventPayload 方法，以 AccelSensor 为例，其参考处理流程如下。

```
void AccelSensor::readEventPayload(EventPayload& payload) {
    ssize_t  n;
    ssize_t  dataAvailable;
    input_event const* event;
    EventPayload::Vec3 vec3 = {                    //初始化默认值
        .x = 0,
        .y = 0,
        .z = -9.8,
        .status = SensorStatus::ACCURACY_HIGH,
    };

    n = mInputReader.fill(this->mInputStreamFd);  //读取数据
    if (n < 0) {
        Sensor::setCheckSelf();
        ALOGE("AceelSensor %s readEvents fill buffer failed\n", mSensorName);
        goto out;
    }
    dataAvailable = mInputReader.readEvent(&event);  //取数据
    if (dataAvailable <= 0) {
        Sensor::setCheckSelf();
        ALOGE("sensor %s no sensor data available\n", mSensorName);
        goto out;
    }

    while (dataAvailable) {
        int  type = event->type;

        if ((type == EV_ABS) || (type == EV_REL) || (type == EV_KEY)) {
            processEvent(event->code, event->value);                    //针对取到的3轴数据进行处理
        } else if (type == EV_SYN) {                                    //sync时表示一次完整完成
            if (reviseEvent()) {
                break;
            }
        }
        dataAvailable = mInputReader.readEvent(&event);                 //继续读取数据
    }

out:
    vec3.x = this->mCurEventData[0];
    vec3.y = this->mCurEventData[1];
    vec3.z = this->mCurEventData[2];
    payload.set<EventPayload::Tag::vec3>(vec3);                         //返回数据
}
```

其中，processEvent 是通常每个 sensor 自行实现，对读取的原始数据进行处理转换的地方。

4 支持 Sensorhal

4.1 编译

在机器上增加 sensorhal 支持，需要在方案中将 sensorhal 编译进去。

```
PRODUCT_PACKAGES += android.hardware.sensors-service.aw
```

在 Android 12 及之前版本，使用 HIDL 2.0 版本。

```
PRODUCT_PACKAGES += android.hardware.sensors@2.0-service
```

4.2 添加 feature

支持一款新 sensor 时，需要添加对应的 feature，否则可能会造成一些 app 无法获取 sensor，或 GMS 测试失败。

android 的 feature 文件以及支持的很好了，我们需要做的是将需要的 feature 文件复制到指定的目录。

1. Android 原生的 feature 文件存放路径。

```
frameworks/native/data/etc
```

里面的 feature 文件。

```
android.hardware.sensor.accelerometer_limited_axes_uncalibrated.xml
android.hardware.sensor.accelerometer_limited_axes.xml
android.hardware.sensor.accelerometer.xml
android.hardware.sensor.ambient_temperature.xml
android.hardware.sensor.assist.xml
android.hardware.sensor.barometer.xml
android.hardware.sensor.compass.xml
android.hardware.sensor.dynamic.head_tracker.xml
android.hardware.sensor.gyroscope_limited_axes_uncalibrated.xml
android.hardware.sensor.gyroscope_limited_axes.xml
android.hardware.sensor.gyroscope.xml
android.hardware.sensor.heading.xml
android.hardware.sensor.heartrate.ecg.xml
android.hardware.sensor.heartrate.fitness.xml
android.hardware.sensor.heartrate.xml
android.hardware.sensor.hifi_sensors.xml
```

```
android.hardware.sensor.hinge_angle.xml
android.hardware.sensor.light.xml
android.hardware.sensor.proximity.xml
android.hardware.sensor.relative_humidity.xml
android.hardware.sensor.stepcounter.xml
android.hardware.sensor.stepdetector.xml
```

2. 方案中复制需要的 feature 文件到指定路径。

```
PRODUCT_COPY_FILES += \
    frameworks/native/data/etc/android.hardware.sensor.accelerometer.xml:$(TARGET_COPY_OUT_VENDOR)/etc/permissions/android.hardware.sensor.accelerometer.xml \
    frameworks/native/data/etc/android.hardware.sensor.compass.xml:$(TARGET_COPY_OUT_VENDOR)/etc/permissions/android.hardware.sensor.compass.xml \
```

图 4-1: sensor_feature

如图所示，当前版型支持了 accelsensor（重力感应）和 compass（磁力计）。

注意在 Android 12 及之前版本，使用的是自定义的 feature 文件，其名字为 sensor_feature.xml，在 Android13 已删除。

⚠ 注意

不支持某类 sensor 时，应将对应的 sensorfeature 删除，避免出现相关问题。

5 适配 Sensor

适配一款新 Sensor 时，通常有 2 种情况。

1. 该 sensor 类型在 Sensorhal 已有默认代码实现。
2. 该 sensor 类型在 Sensorhal 从未适配。

下面针对 2 种情况进行描述

说明

这里适配的 sensor 指驱动已适配完成，能正常上报数据，提供 enable，delay 节点。

5.1 已实现 Sensor

针对已实现的 sensor，以 AccelSensor（重力感应 gsensor）为例。其实现如下。

5.1.1 加载驱动

参考 SensorHal 工作流程-Sensor 驱动加载，在对应方案目录中的 sensor_modules.cfg，添加需要加载的驱动信息。

5.1.2 添加 Sensor 信息

参考 SensorInfo 章节，在 SensorInit.cpp 中添加对应的 Sensor 信息。

说明

在 hidl 版本，则是在 sensorDetect.cpp 中添加对应信息。

5.1.3 数据处理

AccelSensor 通常需要对数据进行转换，当前 Accel 使用的数据转换为如下格式。

```
x = raw_x X priData X revertX
```

其中，priData 为地球标准重力/pri 数据，revertX 指是否需要对 X 进行翻转，该数据通过方向适配而来，会在后面描述。

```
priData = GRAVITY_EARTH / sensorData.priData;
```

5.1.4 AccelSensor 方向适配

AccelSensor 通过用于自动旋转，而贴片的方向是不确定的，因此很多时候还需要对 AccelSensor 的方向进行适配。

5.1.4.1 方向配置文件

在方案目录，还存在如下配置文件。用于记录预定义的方向。

```
device/softwinner/xxx/xxx/input/gsensor.cfg
```

这里用于存放调试好的方向值，每个设备都有五项值，如下所示。

字段	含义
gsensor_name	Gsensor 名称，必须与驱动中设备名相同
gsensor_direct_x	Gsensor x 轴的方向，当定义成 true 时，x 轴取正值，当定义为 false 时，x 轴取负值
gsensor_direct_y	Gsensor y 轴的方向，当定义成 true 时，y 轴取正值，当定义为 false 时，y 轴取负值
gsensor_direct_z	Gsensor z 轴的方向，当定义成 true 时，z 轴取正值，当定义为 false 时，z 轴取负值
gsensor_xy_revert	XY 轴对调，当设为 TRUE 时，x 轴变为原来 y 轴

参考格式例子如下。

```
;-------------------------
; name:sc7a20
;-------------------------
gsensor_name = sc7a20
gsensor_direct_x = false
gsensor_direct_y = false
gsensor_direct_z = false
gsensor_xy_revert = true
```

5.1.4.2 方向配置

方向适配主要有如下配置，针对不同的场景及需求。

1. gsensor.cfg：存在配置好的方向信息。
2. ro.vendor.sf.rotation：Sensor 的旋转角度，可配置为 [0, 90, 180, 270]，通常配置为 0。

3. ro.vendor.gsi_gsen_rotation：针对 GSI 场景下的旋转角度，可配置为 [0, 90, 180, 270]，无 GSI 需求可忽略。

5.1.4.3 方向适配

方向的适配存在几种情况。

1. 有 GSI 需求，且存在 GSI 和量产固件方向不一致的情况。
2. 无 GSI 需求，忽略 ro.vendor.gsi_gsen_rotation 配置即可。

1. 配置前的准备

配置步骤及配置前的准备如下。

1. 已根据《Android_rotation_ 使用指南》进行了屏幕方向的适配。
2. 使用模组前，请确认 gsensor.cfg 文件中是否已经存在模组的方向配置项，若没有，请添加模组的方向配置项，并将各项默认配置为 0。
3. 设置 ro.vendor.sf.rotation 和 ro.vendor.gsi_gsen_rotation 为 0。

2. cfg 配置

gsensor.cfg 该文件存放在机器的/vendor/etc 目录下，当发现方向不正确时，按照如下步骤进行调试。

Gsensor 方向调试说明：

假定机器的长轴为 X 轴，短轴为 Y 轴，垂直方向为 Z 轴。在 android 系统下，可通过专门的 apk 获取 gsensor 的三轴数据，或者通过 dumpsys sensorservice 来获取三轴数据。

首先调试 Z 轴：

第一步观察现象：

机器平放屏幕朝上，获取 gsensor 的轴数据，平放时，z 轴数据正常是接近 9.8(G 的值)，如果 z 轴数据约为-9.8，则 z 轴反了，否则 z 轴正确。

第二步修改 Z 轴为正确方向:

从 gsensor.cfg 配置中找到对应 gsensor 的配置。如果此时该方向 Z 轴向量（gsnesor_direct_z）的值为 false，则需要修改为 true；当为 true，则需要修改为 false。通过 adb shell 将修改后的 gsnesor.cfg 文件 push 到/vendor/etc 下，重启机器，按第一步观察现象。

其次调试 x、y 轴：

第一步观察现象：

首先将机器旋转到 android 显示的 0 度方向，机器垂直于水平面放置，获取 x、y 轴的数据，此时正常情况下，x 轴数据接近为 0，y 轴数据接近 9.8(G 的值)，z 轴接近为 0。如果此时 x 轴接近(-)9.8 的值，y 轴数据接近为 0，则 x、y 轴需交换。

第二步交换 X，Y 方向

从 gsensor.cfg 配置中找到对应 gsensor 的配置。如果此时该 X，Y 轴互换向量（gsensor_xy_revert）的值为 false，则需要修改为 true，当为 true，则需要修改为 false。通过 adb remount 及 adb shell 将修改后的 gsnesor.cfg 文件 push 到/vendor/etc 下，重启机器，按第一步观察现象。

再次调试 X，Y 轴方向：

第一步观察现象：

首先将机器旋转到 android 显示的 0 度方向，机器垂直于水平面放置，获取 y 轴的数据，此时如果 y 轴数据接近正值的 G 值，则 y 轴方向正确，如果接近负值的 G 值，则 y 轴方向相反。

再将机器旋转到 android 显示的 90 度方向，机器垂直于水平面放置，获取 x 轴数据，此时如果 x 轴数据接近正值的 G 值，则 x 轴方向正确，如果接近负值的 G 值，则 x 轴方向相反。

第二步修改 X，Y 轴方向：

当需要修改 X，Y 轴方向时，当只有长轴方向相反或者是只有短轴方向相反时，则只修改方向不正确的一个轴，当两个方向都相反时，则同时修改 X 与 Y 轴方向向量。找到当前使用模组的方向向量（根据模组的名称）。

若长轴方向相反，如果此时该方向 X 轴向量（gsnesor_direct_x）的值为 false，则需要修改为 true，当为 true，则需要修改为 false。

若短轴方向相反，如果此时该方向 Y 轴向量（gsnesor_direct_y）的值为 false，则需要修改为 true，当为 true，则需要修改为 false。

通过 adb shell 将修改后的 gsnesor.cfg 文件 push 到/vendor/etc 下，重启机器，按第一步观察现象。

3. gsi 配置

基于量产固件配置完成 cfg 后且显示正常后，烧录 GSI，然后观察屏幕旋转方向，如旋转正常，则 ro.vendor.gsi_gsen_rotation 配置为 0 即可。

如果旋转异常，可参考如下进行配置。

1. 方向翻转 180°，则配置 ro.vendor.gsi_gsen_rotation=180。
2. 如果方向向左或者向右旋转了 90°，则可尝试配置 ro.vendor.gsi_gsen_rotation 为 90 或者 270 度进行尝试。

5.2 未定义的 sensor 类型

适配一个 SensorHal 中不存在的 sensor，可参考如下步骤。

1. 新建一个 sensor 类型的 cpp，名字可参考 AccelSensor。
2. 实现构造函数
3. 重写 processEvent，readEventPayload 方法，其他方法则看需求重写。
4. 参考已实现 Sensor 适配章节进行适配即可。

6 调试方法

6.1 dumpsys sensorservice

通过 dumpsys sensorservice 可获取相关的 sensor 信息。

主要的信息如下。

1. 当前的 sensor 信息

```
a523-pro:/ # dumpsys sensorservice
Captured at: 13:23:17.866
Sensor Device:
Total 4 h/w sensors, 4 running 0 disabled clients:
Sensor List:
0x00000001) Mi3da 3-axis Accelerometer | miramems Semiconductor Inc. | ver: 1 | type: android.sensor.accelerometer(1)
        | perm: n/a | flags: 0x00000000
            continuous | minRate=1.00Hz | maxRate=100.00Hz | no batching | non-wakeUp |
0x00000002) Mi3da 3-axis Accelerometer uncalibrated | miramems Semiconductor Inc. | ver: 1 | type: android.sensor.
        accelerometer_uncalibrated(35) | perm: n/a | flags: 0x00000000
            continuous | minRate=1.00Hz | maxRate=100.00Hz | no batching | non-wakeUp |
0x00000003) Memsic MMC5603NJ 3-axis Magnetic field sensor | Memsic MMC5603NJ | ver: 1 | type: android.sensor.
        magnetic_field(2) | perm: n/a | flags: 0x00000000
            continuous | minRate=1.00Hz | maxRate=100.00Hz | no batching | non-wakeUp |
0x00000004) Memsic MMC5603NJ 3-axis Magnetic field sensor uncalibrated | Memsic MMC5603NJ | ver: 1 | type: android.
        sensor.magnetic_field_uncalibrated(14) | perm: n/a | flags: 0x00000000
            continuous | minRate=1.00Hz | maxRate=100.00Hz | no batching | non-wakeUp |
0x5f67656f) GeoMag Rotation Vector Sensor | AOSP              |  ver:  3 |  type:  android.sensor.
        geomagnetic_rotation_vector(20) | perm: n/a | flags: 0x00000000
            continuous | maxDelay=0us | maxRate=100.00Hz | no batching | non-wakeUp |
Fusion States:
9-axis  fusion disabled (0 clients),  gyro-rate= 200.00Hz, q=< 0, 0,  0,  0 > (0),  b=< 0, 0,  0 >
game fusion(no mag) disabled (0 clients), gyro-rate= 200.00Hz, q=< 0, 0, 0,  0 > (0),  b=< 0,  0,  0 >
geomag fusion (no gyro) disabled (0 clients),  gyro-rate= 200.00Hz, q=< 0, 0,  0,  0 > (0),  b=< 0, 0,  0 >
```

2. 最近的 50 个 sensor 事件。

```
Recent Sensor events:
Mi3da 3-axis Accelerometer: last 50 events
            1 (ts=449.681903088, wall=11:20:08.027) -0.13, 6.75, 7.10,
            2 (ts=449.746894046, wall=11:20:08.092) -0.13, 6.76, 7.07,
            3 (ts=449.811990255, wall=11:20:08.157) -0.13, 6.80, 7.05,
            4 (ts=449.877012046, wall=11:20:08.222) -0.13, 6.78, 7.05,
            5 (ts=449.877068005, wall=11:20:08.222) -0.15, 6.78, 7.08,
            6 (ts=449.942762088, wall=11:20:08.288) -0.16, 6.79, 7.07,
            7 (ts=450.007137171, wall=11:20:08.352) -0.13, 6.76, 7.10,
```

```
 8 (ts=450.072196380, wall=11:20:08.418) -0.12, 6.78, 7.08,
 9 (ts=450.137091588, wall=11:20:08.482) -0.12, 6.81, 7.08,
10 (ts=450.137176130, wall=11:20:08.482) -0.13, 6.81, 7.02,
```

3. 当前活跃的 sensor。

```
Active sensors:
Socket Buffer size = 39 events
WakeLock Status: not held
Mode : NORMAL
Sensor Privacy: disabled
0 active connections
0 direct connections
```

4. 监听 sensor 的历史进程信息。

```
Previous Registrations:
11:20:10 - 0x00000001 pid= 619 uid= 1000 package=com.android.server.wm.WindowOrientationListener
11:20:00 + 0x00000001 pid= 619 uid= 1000 package=com.android.server.wm.WindowOrientationListener
      samplingPeriod=66667us batchingPeriod=100000us
11:19:59 - 0x00000001 pid= 619 uid= 1000 package=com.android.server.wm.WindowOrientationListener
11:19:29 + 0x00000001 pid= 619 uid= 1000 package=com.android.server.wm.WindowOrientationListener
      samplingPeriod=66667us batchingPeriod=100000us
```

6.2 remonut 后推库

修改代码后，可通过推库的方式替换对应文件，重启观察效果。

```
adb root
adb remount
```

可修改的有。

1. 编译的 bin 文件：vendor/bin/hw/android.hardware.sensors-service.aw。
2. 启动 sensorhal 的 rc 文件：vendor/etc/init/sensors-default-aw.rc。
3. gsensor 方向配置文件：vendor/etc/gsensor.cfg。
4. sensor 模块加载文件：vendor/etc/sensor_modules.cfg。

7 常见问题

7.1 dumpsys sensorservice 能看到设备信息，app 却无法正确获得 sensor

排查是否是 sensorinfo 信息存在错误，如 delay 错误的设置了 0，sensor 类型 flag 设置错误。

1. 通常 delay 不应该设置为 0。
2. flag 针对如 gsensor 等连续上报数据的 sensor，应设置为 SENSOR_FLAG_CONTINUOUS_MODE，如距离感应等，应设置为 SENSOR_FLAG_ON_CHANGE_MODE。

7.2 节点无法写入，提示没有权限

sensor 相关节点，主要有 enable 和 delay，该节点是驱动创建的，当前可通过 uevent 机制修改相关节点权限。

以 mir3da 驱动，节点为 enable 和 delay，适配步骤参考如下。

1. 在/sys/class/input 目录下通过ls -l获取正确的节点路径并查看对应的节点权限信息，其中 input3 为 mir3da 驱动。

```
a523-pro:/sys/class/input # ls -l
total 0
lrwxrwxrwx 1 root root 0 2023-03-16 14:14 event0 -> ../../devices/platform/soc@3000000/7081400.s_twi0/i2c-6/6-0034/axp2101-pek.0/input/input0/event0
lrwxrwxrwx 1 root root 0 2023-03-16 14:14 event1 -> ../../devices/platform/soc@3000000/soc@3000000:codec_mach/sound/card0/input1/event1
lrwxrwxrwx 1 root root 0 2023-03-16 14:14 event2 -> ../../devices/platform/soc@3000000/soc@3000000:daudio0_mach/sound/card1/input2/event2
lrwxrwxrwx 1 root root 0 2023-03-16 14:14 event3 -> ../../devices/virtual/input/input3/event3
lrwxrwxrwx 1 root root 0 2023-03-16 14:14 event4 -> ../../devices/virtual/input/input4/event4
lrwxrwxrwx 1 root root 0 2023-03-16 14:14 event5 -> ../../devices/platform/soc@3000000/2009800.lradc/input/input5/event5
lrwxrwxrwx 1 root root 0 2023-03-16 14:14 event6 -> ../../devices/virtual/input/input6/event6
lrwxrwxrwx 1 root root 0 2023-03-16 14:14 event7 -> ../../devices/virtual/input/input7/event7
lrwxrwxrwx 1 root root 0 2023-03-16 14:14 input0 -> ../../devices/platform/soc@3000000/7081400.s_twi0/i2c-6/6-0034/axp2101-pek.0/input/input0
lrwxrwxrwx 1 root root 0 2023-03-16 14:14 input1 -> ../../devices/platform/soc@3000000/soc@3000000:codec_mach/sound/card0/input1
lrwxrwxrwx 1 root root 0 2023-03-16 14:14 input2 -> ../../devices/platform/soc@3000000/soc@3000000:daudio0_mach/sound/card1/input2
lrwxrwxrwx 1 root root 0 2023-03-16 11:12 input3 -> ../../devices/virtual/input/input3
lrwxrwxrwx 1 root root 0 2023-03-16 14:14 input4 -> ../../devices/virtual/input/input4
lrwxrwxrwx 1 root root 0 2023-03-16 14:14 input5 -> ../../devices/platform/soc@3000000/2009800.lradc/input/input5
lrwxrwxrwx 1 root root 0 2023-03-16 14:14 input6 -> ../../devices/virtual/input/input6
lrwxrwxrwx 1 root root 0 2023-03-16 14:14 input7 -> ../../devices/virtual/input/input7
a523-pro:/sys/class/input #
```

图 7-1: 节点路径

如图所示，可获取到 mir3da 驱动正确的节点路径是/sys/devices/virtual/input/input3。

2. 在方案目录的 uevent.rc 中，添加对应的节点信息，以 Android 13 为例，路径如下。

device/softwinner/saturn/common/system/ueventd.sun55iw3p1.rc

对应添加的 uevent 信息如下。

```
/dev/teepriv0                0776    media       mediadrm
/dev/vipcore                 0666    system      system
/dev/galcore                 0666    system      system
/dev/input/*                  0660    system      input
/dev/sunxi_drm_heap          0777    system      graphics
/dev/graphics/fb0            0666    system      graphics
/dev/deinterlace             0666    media       media

# sysfs properties
/sys/devices/virtual/input/input*   delay          0660  system   system
/sys/devices/virtual/input/input*   enable         0660  system   system
/sys/devices/virtual/input/input*   ps_poll_delay  0660  system   system
/sys/devices/virtual/input/input*   ls_poll_delay  0660  system   system
```

图 7-2: uevent

参考为。

```
/sys/devices/virtual/input/input*   delay          0660  system   system
/sys/devices/virtual/input/input*   enable         0660  system   system
```

其表示为，针对/sys/devices/virtual/input/input* 下的所有 delay 和 enable 节点，修改为 system 用户切权限为 660。

3. 在驱动创建节点位置，上报 uevent 告知节点变动。

```
result = sysfs_create_group(&idev->dev.kobj, &mir3da_attr_group);   //注册节点信息
if (result) {
        MI_ERR("create device file failed!\n");
        result = -EINVAL;
        goto err_unregister_polled_device;
}

kobject_uevent(&idev->dev.kobj, KOBJ_CHANGE);    //固定用法，上报节点变动信息，注意传入参数
```

对于其他名称不同节点，如 ps_poll_delay，方法一致。

著作权声明

本文档及内容受著作权法保护，其著作权由珠海全志科技股份有限公司（“全志”）拥有并保留一切权利。

本文档是全志的原创作品和版权财产，未经全志书面许可，任何单位和个人不得擅自摘抄、复制、修改、发表或传播本文档内容的部分或全部，且不得以任何形式传播。

商标声明





（不完全列举）均为珠海全志科技股份有限公司的商标或者注册商标。在本文档描述的产品中出现的其它商标，产品名称，和服务名称，均由其各自所有人拥有。

免责声明

您购买的产品、服务或特性应受您与珠海全志科技股份有限公司（“全志”）之间签署的商业合同和条款的约束。本文档中描述的全部或部分产品、服务或特性可能不在您所购买或使用的范围内。使用前请认真阅读合同条款和相关说明，并严格遵循本文档的使用说明。您将自行承担任何不当使用行为（包括但不限于如超压，超频，超温使用）造成的不利后果，全志概不负责。

本文档作为使用指导仅供参考。由于产品版本升级或其他原因，本文档内容有可能修改，如有变更，恕不另行通知。全志尽全力在本文档中提供准确的信息，但并不确保内容完全没有错误，因使用本文档而发生损害（包括但不限于间接的、偶然的、特殊的损失）或发生侵犯第三方权利事件，全志概不负责。本文档中的所有陈述、信息和建议并不构成任何明示或暗示的保证或承诺。

本文档未以明示或暗示或其他方式授予全志的任何专利或知识产权。在您实施方案或使用产品的过程中，可能需要获得第三方的权利许可。请您自行向第三方权利人获取相关的许可。全志不承担也不代为支付任何关于获取第三方许可的许可费或版税（专利税）。全志不对您所使用的第三方许可技术做出任何保证、赔偿或承担其他义务。