[Android Sensor传感器系统架构初探](http://blog.csdn.net/qianjin0703/article/details/5942579)

1. 体系结构

2. 数据结构

3. 四大函数

本文以重力感应器装置G-sensor为例探索Android的各层次结构。

**1. 体系结构**

    Android的体系结构可分为4个层次。



* 第一层次  底层驱动层，包括标准Linux，Android核心驱动，Android相关设备驱动，G-sensor的设备驱动程序即存在于此
* 第二层次 Android标准C/C++库，包括硬件抽象层，Android各底层库，本地库，JNI
* 第三层次 Android Java Framwork框架层
* 第四层次 Java应用程序

本文重点关注硬件抽象层，JNI以及Framework。

1.1 硬件抽象层

      硬件抽象层通过例如open(), read(), write(), ioctl(), poll()等函数调用的方式，与底层设备驱动程序进行交互，而这些函数调用是底层设备驱动程序事先准备好的。

      用于交互的关键是文件描述符fd，fd通过open()打开G-sensor设备节点而得到，即 fd = open ("/dev/bma220", O\_RDONLY);而/dev/bma220这个设备节点是在底层设备驱动中注册完成的。

      其他的函数调用如read(), write()等都通过该文件描述符fd对G-sensor设备进行操作。

1.2 JNI (Java Native Interface)

      JNI层可以认为是整个体系结构中的配角，概括地讲，它就完成了一项任务，既实现从C++语言到Java语言的转换。JNI层为Java Framework层提供一系列接口，而这些接口函数的具体实现中，利用例如module->methods->open(), sSensorDevice->data\_open(), sSensorDevice->poll()等回调函数与硬件抽象层进行交互。而这些open(), poll()回调函数在硬件抽象层中具体实现。

1.3 Java Framework

      Framework层提供各种类和类的对象，可作为系统的守护进程运行，也可供上层应用程序的使用。

      例如类SensorManager，它作为系统的守护进程在初始化的时候开始运行，其子类SensorThread中的子类SensorThreadRunnable通过sensors\_data\_poll()实现了对G-sensor数据的轮训访问，而sensors\_data\_poll()通过JNI层转换到硬件抽象层去具体实现poll()。

**2 数据结构**

     一般境况下，硬件抽象层对硬件的描述都分为control和data两大类。

2.1 sensors\_control\_context\_t

struct sensors\_control\_context\_t {  
    struct sensors\_control\_device\_t device;

    int fd;  
};

struct sensors\_control\_device\_t {  
    struct hw\_device\_t common;  
    int (\***open\_data\_source**)(struct sensors\_control\_device\_t \*dev);  
    int (\*activate)(struct sensors\_control\_device\_t \*dev, int handle, int enabled);  
    int (\*set\_delay)(struct sensors\_control\_device\_t \*dev, int32\_t ms);

    int (\*wake)(struct sensors\_control\_device\_t \*dev);  
};

2.2 sensors\_data\_context\_t

struct sensors\_data\_context\_t {  
    struct sensors\_data\_device\_t device;

    int fd;  
};

struct sensors\_data\_device\_t {  
    struct hw\_device\_t common;

    int (\***data\_open**)(struct sensors\_data\_device\_t \*dev, int fd);  
    int (\*data\_close)(struct sensors\_data\_device\_t \*dev);  
    int (\***poll**)(struct sensors\_data\_device\_t \*dev,  
            sensors\_data\_t\* data);  
}

struct hw\_device\_t {  
    uint32\_t tag; uint32\_t version;

    struct hw\_module\_t\* module;

    int (\*close)(struct hw\_device\_t\* device);  
};

struct hw\_module\_t {  
    uint32\_t tag; uint16\_t version\_major; uint16\_t version\_minor;

    const char \*id; const char \*name; const char \*author;

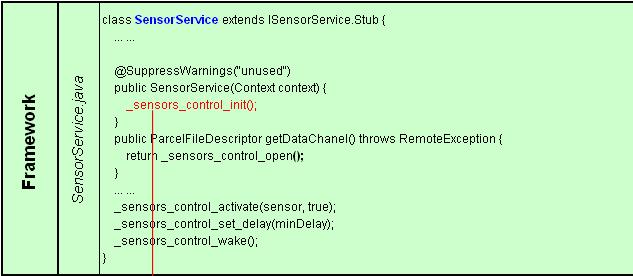
    struct hw\_module\_methods\_t\* methods;  
};

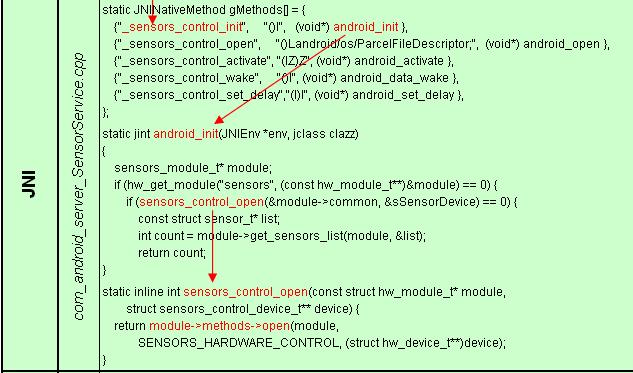
struct hw\_module\_methods\_t {  
    int (\***open**)(const struct hw\_module\_t\* module, const char\* id,  
            struct hw\_device\_t\*\* device);  
};

下文将通过对(\*open), (\*open\_data\_source), (\*data\_open)和(\*poll)的代码分析，探索Android的各层次架构。

**3 四大函数**

3.1 module->methods->open()







1) Framework

    SensorService作为系统守护进程运行，其类的构造函数实现\_sensors\_control\_init()。

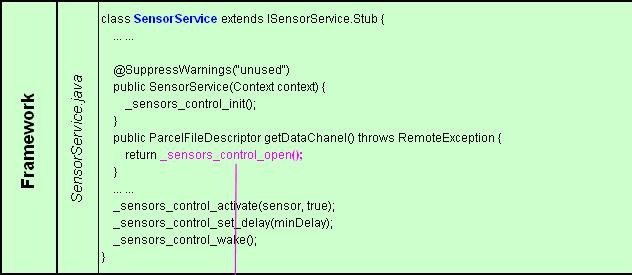
2) JNI

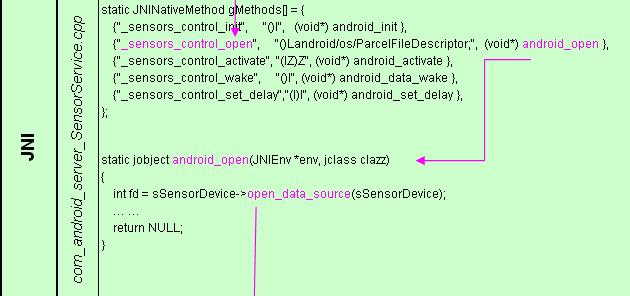
    为\_sensors\_control\_init()提供接口android\_init()，并执行回调函数module->methods->open();

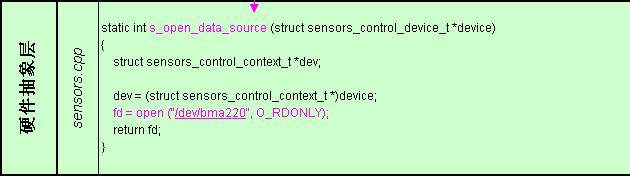
3) 硬件抽象层

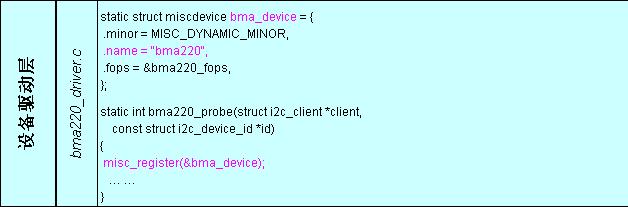
    具体实现(\*open)，该函数为所有G-sensor回调函数的指针赋值。

3.2 sSensorDevice->open\_data\_source()









1) Framework

    SensorService作为系统守护进程运行，其类的一个公有成员ParcelFileDescriptor通过实现\_sensors\_control\_open()得到设备的文件描述符。

2) JNI

    为\_sensors\_control\_open()提供接口android\_open()，并执行回调函数sSensorDevice->open\_data\_source();

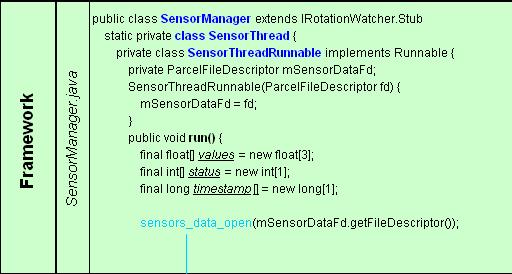
3) 硬件抽象层

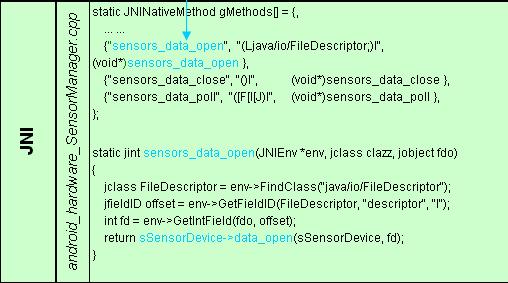
    具体实现(\*open\_data\_source)，该函数通过打开G-sensor的设备节点得到文件描述符fd = open ("/dev/bma220", O\_RDONLY);

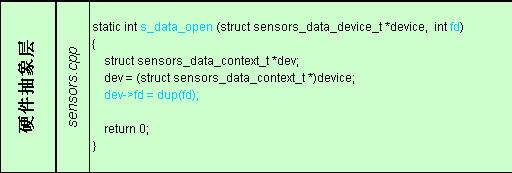
4) 设备驱动层

    通过misc\_register()对G-sensor设备进行注册，建立设备节点。

3.3 sSensorDevice->data\_open()







1) Framework

    SensorManager作为系统守护进程运行，其子类SensorThreadRunnable的行为函数run()实现sensors\_data\_open()。

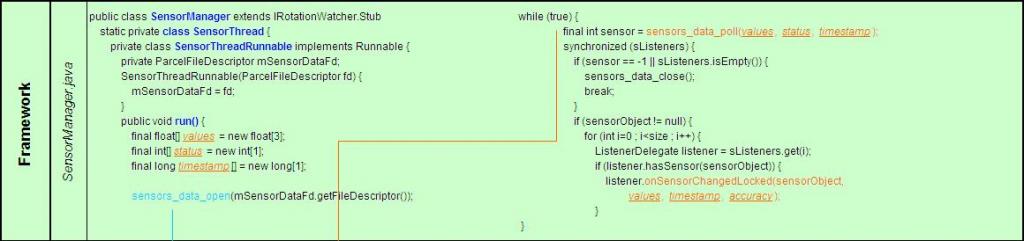
2) JNI

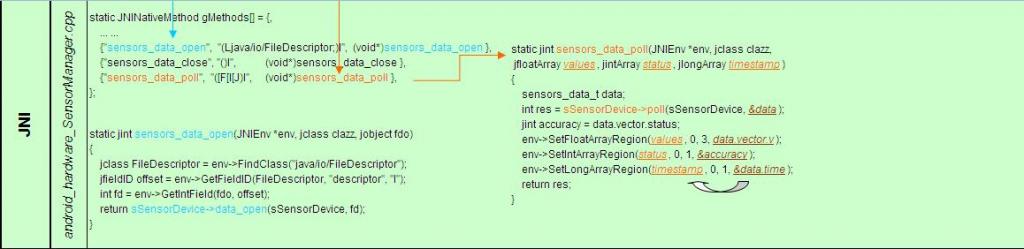
    为sensors\_data\_open()提供接口sensors\_data\_open()，并执行回调函数sSensorDevice->data\_open();

3) 硬件抽象层

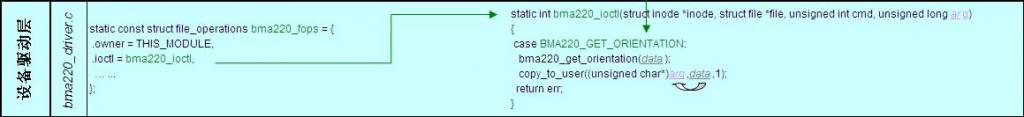
    具体实现(\*data\_open)，该函数的功能就是将已经得到的文件描述符fd复制一份到sensors\_data\_context结构体中的dev->fd，以便为处理数据的回调函数如(\*poll)使用。

3.4 sSensorDevice->poll()







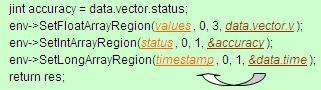


1) Framework

    SensorManager作为系统守护进程运行，其子类SensorThreadRunnable的行为函数run()实现sensors\_data\_poll(values, status, timestamp)，其目的是通过此函数得到从底层传上来的有关G-sensor的数据values, status和timestamp，再通过此类的一个行为函数listener.onSensorChangedLocked(sensorObject, values, timestamp, accuracy);为上层应用程序提供了得到G-sensor设备数据的接口函数。

2) JNI

    为sensors\_data\_poll()提供接口sensors\_data\_poll()，并执行回调函数sSensorDevice->poll(sSensorDevice, &data);其中，得到的data就是从底层传上来的G-sensor数据，然后通过下图的方式将data中对应的数据分别赋给values, status和timestamp。

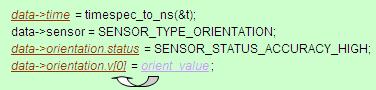


3) 硬件抽象层

    具体实现(\*poll)，该函数通过ioctl()实现与底层驱动程序的交互。

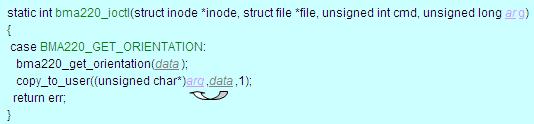
    ioctl(dev->fd, BMA220\_GET\_ORIENTATION, &orient\_value);

    其中，dev->fd即刚才由(\*data\_open)得到的文件描述符，BMA220\_GET\_ORIENTATION为ioctl的一个命令，具体实现由底层驱动程序完成，orient\_value即得到的G-sensor数据，它通过下图的方式将相对应的数据赋给了data结构体中的values, status和time，从而最终实现了从底层到上层的数据通信。



4) 设备驱动层

    与硬件抽象层交互的read(), write(), ioctl()函数由设备驱动实现。以ioctl()的一条命令BMA220\_GET\_ORIENTATION为例，



通过bma220\_get\_orientation(data)得到G-sensor的数据data，然后将其从内核空间上传到用户空间的arg.