android上层应用apk到G-sensor driver的大致流程：

Android对于Sensor的API定义在 hardware/libhardware/include/hardware/sensor.h中, 要求在sensor.so提供以下8个API函数

[控制方面]

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);

[数据方面]

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);

[模块方面]

int (\*get\_sensors\_list)(struct sensors\_module\_t\* module, struct sensor\_t const\*\* list);

在**Java层**Sensor的**状态控制**由**SensorService来负责**,它的java代码和JNI代码分别位于:

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

frameworks/base/services/jni/com\_android\_server\_SensorService.cpp

在**Java层**Sensor的**数据控制**由**SensorManager来负责**,它的java代码和JNI代码分别位于:

frameworks/base/core/java/android/hardware/SensorManager.java

frameworks/base/core/jni/android\_hardware\_SensorManager.cpp

android framework中与sensor通信的是sensorService.java和sensorManager.java。

sensorService.java的具体通信是通过JNI调用sensorService.cpp中的方法实现的。

sensorManager.java的具体通信是通过JNI调用sensorManager.cpp中的方法实现的。

sensorService.cpp和sensorManger.cpp通过hardware.c与sensor.so通信。其中sensorService.cpp实现对sensor的状态控制，sensorManger.cpp实现对sensor的数据控制。

sensor.so通过ioctl控制sensor driver的状态，通过打开sensor driver对应的设备文件读取G-sensor采集的数据。

android SDK提供了4个类来于sensor通信，分别为 sensor,sensorEvent,sensorEventListener,sensorManager.其中 sensorEventListener用来在sensorManager中注册需要监听的sensor类型。

sensorManager.java提供registrater(),unregistrater()接口供sensorEventListener使用。

sensorManager.java不断轮询从sensor.so中取数据。取到数据后送给负责监听此类型sensor的 sensorEventListener.java。sensorEventListener.java通过在sensorManager.java中注 册可以监听特定类型的sensor传来的数据。

系统启动时执行systemProcess,会启动sensorService.java,在sensorService.java的构造函数中调用JNI方法\_sensor\_control\_init()。

sensorService.cpp中相应的方法android\_int()会被执行。该函数会调用hardware.c中的方法hw\_get\_module()此函数又通过调用load()函数在system/lib/hw下查找sensor.so

查找时会根据harware.c中定义好的sensor.\*.so的扩展名的顺序查找，找到第一个匹配的时候即停止，并将该sensor.so中定义好的一个全局变量HAL\_MODULE\_INFO\_SYM带回。该变量包含的一个

重要信息是它的一个成员结构变量中包含的一个函数指针open，该指针所指函数会对一个device结构变量赋值，从而带出sensorService.cpp 和sensorManager.cpp与sensor通信所需要的全部信息。

device结构变量有两种变体分别供sensorService.cpp和sensorManaer.cpp使用。其中主要是一些函数指针指向与sensor通信的函数。

sensorService.cpp和sensorManager.cpp在得到HAL\_MODULE\_INFO\_SYM结构后都会调用 sensors.h的inline函数open()通过HAL\_MODULE\_INFO\_SYM的open函数指针将所需的device信息取回。

系统在启动activityManager.java时，它会启动sensorManager.java，它也会调用hardware.c中的方法hw\_get\_module()带回HAL\_MODULE\_INFO\_SYM。