设备中的三自由度陀螺仪就是一个可以识别设备相对于地面，绕x、y、z轴转动角度的感应器。智能手机，平板电脑有了它，可以实现很多好玩的应用，比如说指南针等。  
  
        我们可以用一个磁场感应器(magnetic sensor)，来实现陀螺仪。  
  
        磁场感应器是用来测量磁场感应强度的。一个3轴的磁sensor IC可以得到当前环境下X、Y和Z方向上的磁场感应强度，对于Android中间层来说就是读取该感应器测量到的这3个值。当需要时，上报给上层应用程序。磁感应强度的单位是T（特斯拉）或者是Gs（高斯），1T等于10000Gs。  
  
       了解陀螺仪前先来看看android定义的坐标系，在/hardware/libhardware/include/hardware/sensors.h中有个图。

       图中表示设备的正上方是y轴方向，右边是x轴方向，垂直设备屏幕平面向上的是Z轴方向，这个很重要。因为应用程序就是根据这样的定义来写的，所以我们报给应用的数据要跟这个定义符合。还需要清楚磁sensor芯片贴在板上的坐标系。我们从芯片读出数据后要把芯片的坐标系转换为设备的实际坐标系。除非芯片贴在板上刚好跟设备的x、y、z轴方向刚好一致（去感谢你的硬件工程师吧）。

       陀螺仪的实现是根据磁场感应强度的3个值计算出另外3个值。当需要时，我们计算出这3个值上报给应用程序，陀螺仪的功能就实现了。

        sensors.h中还定义了其他各种sensor。要实现的就是这两个：

        #define SENSOR\_TYPE\_MAGNETIC\_FIELD      2

        #define SENSOR\_TYPE\_ORIENTATION         3

        在/hardware/sensors/sensors.cpp 中添加对MAGNETIC\_FIELD和ORIENTATION 的支持，简单的说一下怎样添加,下面的代码不完整，请参考/sdk/emulator/sensors/sensors\_qemu.c

**java代码：**

1. //加入需要的宏定义
2. #define ID\_BASE SENSORS\_HANDLE\_BASE
3. #define ID\_ACCELERATION (ID\_BASE+0)
4. #define ID\_MAGNETIC\_FIELD (ID\_BASE+1)
5. #define ID\_ORIENTATION (ID\_BASE+2)
6. #define S\_HANDLE\_ACCELEROMETER (1<<ID\_ACCELERATION)
7. #define S\_HANDLE\_MAGNETIC\_FIELD (1<<ID\_MAGNETIC\_FIELD)
8. #define S\_HANDLE\_ORIENTATION (1<<ID\_ORIENTATION)
9. #define SENSORS\_NUM 4
10. #define SUPPORTED\_SENSORS ((1<<NUM\_SENSORS)-1)
11. //在 sensor\_t sensors\_list[] 中添加两个sensor的信息,
12. //这些只是一些Sensor的信息，应用程序可以获取到。
13. #ifdef MAGNETIC\_FIELD { name : "XXX 3-axis Magnetic field sensor", vendor : "XXX company", version : 1, handle : S\_HANDLE\_MAGNETIC\_FIELD, type : SENSOR\_TYPE\_MAGNETIC\_FIELD, maxRange : 600.0f,
14. //最大范围 resolution : 30.0f,
15. //最小分辨率 power : 6.7f,
16. //这个不太懂
17. },
18. #endif
19. #ifdef ORIENTATION {
20. name: "XXX Orientation sensor", vendor: "XXX company", version: 1, handle: S\_HANDLE\_ORIENTATION, type: SENSOR\_TYPE\_ORIENTATION, maxRange: 360, resolution: 0.1, power: 20, },
21. #endif
22. //定义一个结构来保存orientation的信息
23. static struct orientation{
24. float azimuth; float pitch; float roll;
25. }orientation;
26. //在 control\_\_open\_data\_source()函数中打开设备
27. static native\_handle\_t\* control\_\_open\_data\_source(struct sensors\_control\_device\_t \*dev) { SensorControl\* ctl = (void\*)dev;
28. native\_handle\_t\* handle;
29. int fd\_m = open (MAGNETIC\_DATA\_DEVICE, O\_RDONLY);
30. LOGD ("Open Magnetic Data source: %d, %d/n", fd\_m, errno);
31. if (fd\_m>= 0) {
32. dev->fd[ID\_MAGNETIC\_FIELD] = dup(fd\_m);
33. }
34. return handle;
35. }
36. //实现数据的打开和关闭函数
37. static int data\_\_data\_open(struct sensors\_data\_device\_t \*dev, native\_handle\_t\* handle) {
38. struct sensors\_data\_context\_t \*dev;
39. dev = (struct sensors\_data\_context\_t \*)device;
40. for(int i=0 ;i<SENSORS\_NUM; i++) {
41. dev->fd[i] = dup(handle->data[i]);
42. }
43. native\_handle\_close(handle);
44. native\_handle\_delete(handle);
45. return 0;
46. }
47. static int data\_\_data\_close(struct sensors\_data\_device\_t \*dev) {
48. struct sensors\_data\_context\_t \*dev;
49. dev = (struct sensors\_data\_context\_t \*)device;
50. for(int i=0 ;i<SENSORS\_NUM; i++) {
51. if (dev->fd[i] >= 0) {
52. close(dev->fd[i]);
53. }
54. dev->fd[i] = -1;
55. }
56. return 0;
57. }
58. //最关键的poll函数
59. static int data\_\_poll(struct sensors\_data\_device\_t \*dev, sensors\_data\_t\* values) {
60. SensorData\* data = (void\*)dev; int fd = data->events\_fd;
61. //判断设备是否打开
62. if(dev->fd[ID\_MAGNETIC\_FIELD] < 0) {
63. LOGD("In %s dev[%d] is not open!/n",\_\_FUNCTION\_\_ ,ID\_MAGNETIC\_FIELD);
64. return -1;
65. }
66. pollfd pfd[SENSORS\_NUM] = {
67. //省略其他sensor代码 {
68. fd: dev->fd[ID\_MAGNETIC\_FIELD], events: POLLIN, revents: 0
69. },
70. //省略其他sensor代码
71. };
72. int err = poll (pfd, SENSORS\_NUM, s\_timeout);
73. unsigned int mask = SUPPORTED\_SENSORS;
74. static unsigned int poll\_flag=0;
75. if(poll\_flag==0) {
76. poll\_flag = mask;
77. }
78. //省略其他sensor
79. if(poll\_flag&(1<<ID\_MAGNETIC\_FIELD)) {
80. if((pfd[ID\_MAGNETIC\_FIELD].revents&POLLIN) == POLLIN) {
81. char rawData[6];
82. err = read (dev->fd[ID\_MAGNETIC\_FIELD], &rawData, sizeof(rawData));
83. if(err<0) {
84. LOGE("read magnetic field ret:%d errno:%d/n", err, errno);
85. return err;
86. }
87. struct timespec t;
88. clock\_gettime(CLOCK\_REALTIME, &t);
89. data->time = timespec\_to\_ns(&t);
90. data->sensor = SENSOR\_TYPE\_MAGNETIC\_FIELD;
91. data->magnetic.status = SENSOR\_STATUS\_ACCURACY\_HIGH;
92. //上报的数据单位要转换成
93. uTesla data->magnetic.x = ( (rawData[1] << 8 ) | rawData[0])/ MAGNETIC\_CONVERT; data->magnetic.y = ( (rawData[3] << 8 ) | rawData[2])/ MAGNETIC\_CONVERT; data->magnetic.z = ( (rawData[5] << 8 ) | rawData[4])/ MAGNETIC\_CONVERT;
94. //把陀螺仪需要的数据计算出来，用atan2()，头文件要加上
95. #include <math.h> float azimuth = atan2( (float)(data->magnetic.x ),(float)(data->magnetic.y) );
96. if(azimuth<0) { azimuth = 360 - fabs(azimuth\*180/PI);
97. } else {
98. azimuth = azimuth\*180/PI;
99. }
100. orientation.azimuth = 360-azimuth;
101. //rotation around the X axis.+180~-180 degree orientation.pitch = atan2( (float)(data->magnetic.y ),(float)(data->magnetic.z) )\*180/PI;
102. //rotation around the Y axis +90~-90 degree float roll = atan2( (float)(data->magnetic.x ),(float)(data->magnetic.z) ) \*180/PI;
103. if (roll > 90) {
104. roll = -(180.0-roll);
105. } else if (roll < -90) {
106. roll = 180 + roll;
107. }
108. orientation.roll = roll;
109. }
110. return S\_HANDLE\_MAGNETIC\_FIELD;
111. }
112. if(poll\_flag&(1<<ID\_MAGNETIC\_FIELD)) {
113. //数据已经计算好了直接上报就行
114. struct timespec t;
115. clock\_gettime(CLOCK\_REALTIME, &t);
116. data->time = timespec\_to\_ns(&t);
117. data->sensor = SENSOR\_TYPE\_ORIENTATION;
118. data->orientation.azimuth = orientation.azimuth;
119. data->orientation.pitch = orientation.pitch;
120. data->orientation.roll = orientation.roll;
121. poll\_flag &= ~(1<<ID\_ORIENTATION);
122. return S\_HANDLE\_ORIENTATION;
123. }
124. }