10

感應器應用

10-1 感應器介紹

感應器（sensor）就是專門感應外界事物變化，並將其變化轉爲數值的一種接收器。日常生活中常見的感應器有：溫度計（感應外界溫度變化）、指北針（感應南北極磁場）。另外受歡迎的電視遊樂器 Wii，其搖桿內藏加速度感應器，可以讓 Wii 透過該感應器知道搖桿傾斜的狀況來作適當的回應。

與 Android 感應器有關的函式庫都放在「android.hardware」套件內，目前所支援的感應器如表 10-1 所示[1]：

▼表 10-1

感應器	對應的值
加速度感應器	Sensor.TYPE_ACCELEROMETER
重力感應器	Sensor.TYPE_GRAVITY
陀螺儀感應器	Sensor.TYPE_GYROSCOPE
光線感應器	Sensor.TYPE_LIGHT
線性加速度感應器	Sensor.TYPE_LINEAR_ACCELERATION
磁場感應器	Sensor.TYPE_MAGNETIC_FIELD
方位感應器	SensorManager.getOrientation() 已取代 Sensor.TYPE_ORIENTATION
壓力感應器	Sensor.TYPE_PRESSURE
接近感應器	Sensor.TYPE_PROXIMITY
旋轉向量感應器	Sensor.TYPE_ROTATION_VECTOR
溫度感應器	Sensor.TYPE_TEMPERATURE

[1] 請參看 Android 2.3 版 API 文件關於 Sensor 類別常數的說明。

不可不知

1. 不是每一台 Android 行動裝置都有表 10-1 所列的感應器，如果沒有對應的感應器，即使寫程式也無法取得對應的資料。
2. Android 模擬器無法模擬感應器功能，雖然網路上有模擬感應器的軟體[2]，但還是強烈建議直接在實機上測試方爲上策。

不論是何種感應器，最重要的就是取得其對外界感應後所蒐集到的數值，數值是以一個 float 陣列儲存，通常以 values[i]來代表（values 代表該陣列名稱，i 代表索引），依照不同的感應器，陣列的元素個數也會有所不同。例如加速度與方位感應器都有 X 軸、Y 軸、Z 軸觀念，所以有 3 個數值：values[0]、values[1]、values[2] 以儲存對應資訊；而接近感應器只有距離一個數值，所以只使用到 values[0]。每個數值代表的意義將於下列各個感應器小節再做詳細說明。一般 Android 行動裝置大部分都有加速度感應器、方位感應器、接近感應器與光線感應器的功能，所以本章專門探討這 4 個感應器。雖然未提及其他感應器，但取得數值的方式皆相同。

10-2 加速度感應器

在說明加速度感應器之前，先說明 X 軸、Y 軸、Z 軸所代表的位置。圖 10-1 屬於 3D 座標圖，Android 採用 OpenGL ES 的座標系統，說明如下：

1. 螢幕左下角頂點爲原點（x=0, y=0, z=0），此與一般 2D 座標系統原點在螢幕左上角不同。
2. X 軸爲左向右的水平方向，所以向右 X 值增加，向左 X 值減少。
3. Y 軸爲下向上的垂直方向，所以向上 Y 值增加，向下 Y 值減少。
4. Z 軸爲後向前的方向，所以向前 Z 值增加，向後 Z 值減少。

[2] 請參看 http://www.openintents.org/en/node/23

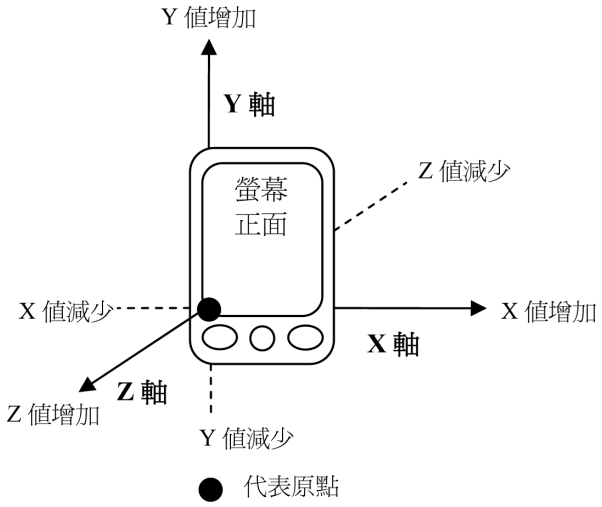

圖 10-1

接下來說明加速度感應器。加速度的單位是 m/sec^2（公尺/秒的平方），而加速度感應器則是反應 X 軸、Y 軸、Z 軸受到地心引力的影響情形，重力方向恰與座標方向相反，所以若符合重力方向與座標方向相反，會得到正的值，反之會得到負的值。如圖 10-2：

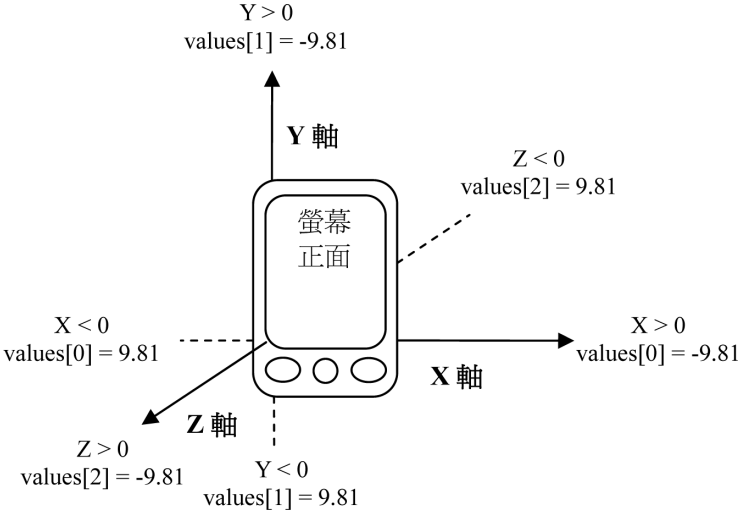

圖 10-2

各種狀態說明如下：

1. 行動裝置平躺（螢幕正面朝上），如圖 10-3，此時 Z 軸受重力影響，values 值如下：
   - values[0] = 0.0，代表 X 軸未受重力影響。
   - values[1] = 0.0，代表 Y 軸未受重力影響。
   - values[2] = 9.81，值爲正代表 Z 軸後面方向（Z < 0）受重力影響。

   若行動裝置平躺但螢幕正面朝下，背蓋朝上，如圖 10-4，則 values[2] = -9.81，代表 Z 軸前面方向受重力影響。

圖 10-3

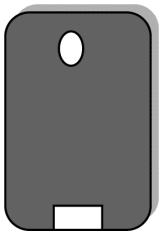

圖 10-4

2. 行動裝置呈現如圖 10-5 的縱向直立狀態（稱爲 portrait），此時 Y 軸受重力影響，values 值如下：
   - values[0] = 0.0，代表 X 軸未受重力影響。
   - values[1] = 9.81，值爲正代表 Y 軸下面方向（Y < 0）受重力影響。
   - values[2] = 0.0，代表 Z 軸未受重力影響。

   若行動裝置縱向直立方式上下顛倒，如圖 10-6，則 values[1] = -9.81，代表 Y 軸上面方向受重力影響。

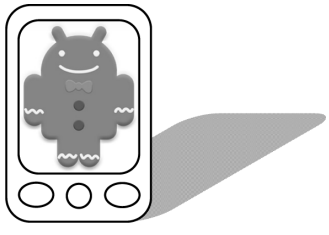

圖 10-5

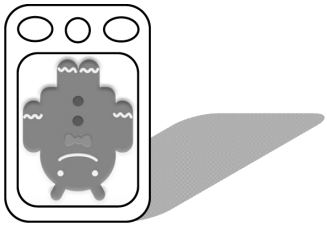

圖 10-6

3. 行動裝置呈現如圖 10-7 的橫向直立狀態（稱爲 landscape），此時 X 軸受重力影響，values 值如下：
   - values[0] = 9.81，代表 X 軸左面方向（X < 0）受重力影響。
   - values[1] = 0.0，代表 Y 軸未受重力影響。
   - values[2] = 0.0，代表 Z 軸未受重力影響。

   若行動裝置橫向直立方式左右顛倒，如圖 10-8，則 values[0] = -9.81，代表 X 軸右面方向受重力影響。

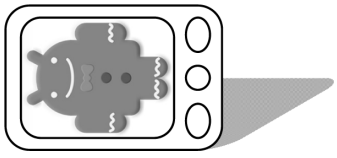

圖 10-7

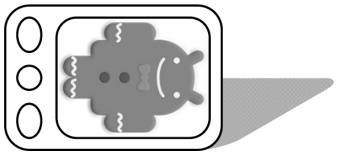

圖 10-8

依照下列步驟可以取得感應器的相關資訊與該感應器對外界感應後所得到的對應數值：

STEP 1 取得 SensorManager 物件：透過 SensorManager 物件方能取得各種感應器的資訊，而要取得該物件必須呼叫 Context 的 getSystemService()，並指定欲取得的系統服務名稱，這點跟取得 NotificationManager、LocationManager 物件方式相同[3]。

```
SensorManager sensorMgr = (SensorManager)getSystemService(SENSOR_SERVICE);
```

STEP 2 實作 SensorEventListener：實作 SensorEventListener 的 onSensorChanged()，當感應器的值改變時會自動呼叫此方法，並傳入 SensorEvent 物件，透過該物件可以取得產生事件的感應器。

```
class MySensorEventListener implements SensorEventListener{
    public void onSensorChanged(SensorEvent event) {
```

[3] 欲知目前有哪些系統服務，請參看 API 文件 Context 的 getSystemService() 的說明。

```
        float[] sensorsValues = event.values; //感應器對外界感應後所蒐集到的數值
        Sensor sensor = event.sensor; //取得產生此事件的感應器
        String sensorName = sensor.getName(); //取得感應器名稱
        int sensorType = sensor.getType(); //取得感應器種類
        float sensorPower = sensor.getPower(); //取得感應器的耗電量
    }

    public void onAccuracyChanged(Sensor sensor, int accuracy) {
        //當感應器的精準度改變時會呼叫此方法
    }
}
```

STEP 3 為指定的感應器註冊 SensorEventListener：呼叫 SensorManager 的 registerListener() 替指定的感應器註冊 SensorEventListener，當感應器的值變化時，SensorEventListener 的 onSensorChanged() 才會自動被呼叫。

```
registerListener(listener, sensor, int rate)
```

取得感應器相關資訊所需使用到的相關方法，說明如表 10-2：

▼表 10-2

Context 類別
public Object getSystemService (String name) 依據指定名稱取得對應系統服務的管理物件。 • name：欲取得的系統服務名稱。
SensorEventListener 介面
public abstract void onSensorChanged (SensorEvent event) 當感應器的值改變時會呼叫此方法。 • event：SensorEvent 物件，透過該物件可以取得感應器相關資訊。
public abstract void onAccuracyChanged (Sensor sensor, int accuracy) 當感應器的精準度改變時會呼叫此方法。 • sensor：產生此事件的感應器。 • accuracy：感應器新的精準度。

SensorEvent 類別
public Sensor sensor 產生事件的對應感應器。
public final float[] values 感應器感知外界環境而蒐集的數值，不同感應器的數值與其代表的意義會不同。
Sensor 類別
public String getName () 取得感應器名稱。
public int getType () 取得感應器種類。
public float getPower () 取得感應器的耗電量。
public float getMaximumRange () 取得感應器可偵測的最大範圍。
SensorManager 類別
public Sensor getDefaultSensor (int type) 依據指定的感應器種類回傳對應的 Sensor 物件。 • type：Sensor 種類。
public boolean registerListener (SensorEventListener listener, Sensor sensor, int rate) 為指定的感應器註冊 SensorEventListener。如果行動裝置有對應的感應器而且可以正常運作回傳 true；否則回傳 false。 • listener：實作 SensorEventListener 的物件。 • sensor：欲註冊的感應器。 • rate：設定事件發生後傳送數值的頻率，有下列幾種（依照頻率低到高排列）： ◆ SENSOR_DELAY_NORMAL－適合螢幕的頻率。 ◆ SENSOR_DELAY_UI－適合使用者介面的頻率 ◆ SENSOR_DELAY_GAME－適合遊戲的頻率。 ◆ SENSOR_DELAY_FASTEST－頻率最高。

SensorManager 類別
public void unregisterListener (SensorEventListener listener) 解除所有對 SensorEventListener 註冊的感應器。 • listener：實作 SensorEventListener 的物件。
public void unregisterListener (SensorEventListener listener, Sensor sensor) 解除對 SensorEventListener 註冊的特定感應器。 • listener：實作 SensorEventListener 的物件。 • sensor：欲解除註冊的感應器。

範例 AccelerometerEx

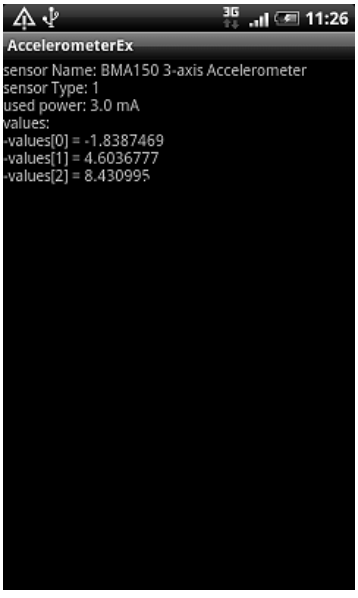

圖 10-9

範例說明：

- 顯示感應器名稱、種類與耗電量。
- 將加速度感應器對外界感應後所得到的數值顯示在畫面上。

- values[0]、values[1]、values[2] 分別反應行動裝置 X 軸、Y 軸、Z 軸受到地心引力影響的情形；值都介於-9.81 ~ +9.81 之間。

AccelerometerEx/src/org/accelerometerEx/AccelerometerEx.java

```
15.    @Override
16.    public void onCreate(Bundle savedInstanceState) {
17.        super.onCreate(savedInstanceState);
18.        setContentView(R.layout.main);
19.        sensorMgr = (SensorManager)getSystemService(SENSOR_SERVICE);
20.        findViews();
21.    }
22.
23.    private void findViews() {
24.        tvMsg = (TextView)findViewById(R.id.tvMsg);
25.    }
26.
27.    SensorEventListener listener = new SensorEventListener() {
28.        public void onSensorChanged(SensorEvent event) {
29.            Sensor sensor = event.sensor;
30.            StringBuilder sensorInfo = new StringBuilder();
31.            sensorInfo.append("sensor Name: " + sensor.getName() + "\n");
32.            sensorInfo.append("sensor Type: " + sensor.getType() + "\n");
33.            sensorInfo.append("used power: " + sensor.getPower() + " mA\n");
34.            sensorInfo.append("values: \n");
35.            float[] values = event.values;
36.            for (int i = 0; i < values.length; i++)
37.                sensorInfo.append("-values[" + i + "] = " + values[i] + "\n");
38.            tvMsg.setText(sensorInfo);
39.        }
40.
41.        public void onAccuracyChanged(Sensor sensor, int accuracy) {
42.            //當感應器的精準度改變時會呼叫此方法
43.        }
44.    };
45.
46.    @Override
47.    protected void onResume() {
48.        super.onResume();
49.        sensorMgr.registerListener(listener,
50.                    sensorMgr.getDefaultSensor(Sensor.TYPE_ACCELEROMETER),
51.                    SensorManager.SENSOR_DELAY_UI);
52.    }
53.
```

```
54.    @Override
55.    protected void onPause() {
56.        super.onPause();
57.        sensorMgr.unregisterListener(listener);
58.    }
```

19 行：呼叫 getSystemService()，並指定「SENSOR_SERVICE」系統服務名稱即可取得 SensorManager 物件。

27-28 行：以匿名內部類別實作 SensorEventListener 的 onSensorChanged()，當感應器的值改變時會自動呼叫此方法。

29 行：取得感應器物件。

31 行：取得感應器名稱。

32 行：取得感應器種類。

33 行：取得感應器耗電量。

35-37 行：取得感應器對外界感應後所蒐集到的數值，並以 for-each 迴圈將對應的值取出。

47-52 行：在操作畫面顯示之前，先將加速度感應器註冊對應的 SensorEventListener，並設定感應器的傳送頻率為「SENSOR_DELAY_UI」。

57 行：在操作畫面消失之前，解除所有對 SensorEventListener 註冊的感應器，以節省電力。

10-3 方位感應器

依據加速度感應器的數值只能判斷受重力影響的方向，比較無法精準判斷是如何旋轉，如圖 10-10，平躺且螢幕正面朝天的行動裝置，無論順時針或逆時針翻轉行動裝置，values[2] 值的變化都是 9.81→ 0，所以無法從加速度感應器的數值解析出是以哪種方式翻轉行動裝置，當然也就無法更精準反應使用者的操作。如果有方位資訊，就能得知行動裝置目前方位，更進一步可以判斷出翻轉方式。

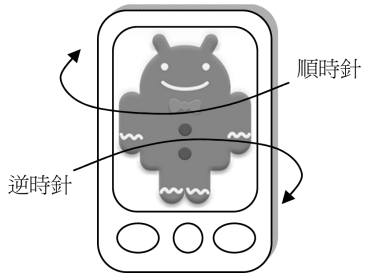

圖 10-10

早期使用方位感應器只要將感應器指定爲「Sensor.TYPE_ORIENTATION」即可。之後 Android 則將「Sensor.TYPE_ORIENTATION」常數設定爲 deprecated，而建議改以呼叫 SensorManager.getOrientation() 方式取得方位感應器的資訊。因爲前者觀念比較直覺容易；而後者則較爲複雜難懂，所以不僅坊間有許多教學書仍然只介紹前者概念，就連許多開發人員亦只以前者方式開發應用程式。本文會將這 2 種取值、解讀數值方式加以比較與說明，讓讀者更容易瞭解它們的差異。因爲 Android 的 API 文件已不建議採用前者，所以本文會先介紹後者，然後再說明前者概念。

10-3-1 呼叫 getOrientation() 取得方位資訊

使用此種方式是透過加速度感應器的數值計算出行動裝置的方位資訊，所以必須先取得加速度感應器的數值。依照下列步驟可以取得方位數值：

STEP 1 取得加速度感應器與磁場感應器的數值（所以必須將加速度感應器與磁場感應器註冊對應的 SensorEventListener）。

```
public void onSensorChanged(SensorEvent event) {
    switch (event.sensor.getType()) {
        case Sensor.TYPE_ACCELEROMETER:
            //取得加速度感應器的數值
            accelerometer_values = (float[]) event.values.clone();
            break;
        case Sensor.TYPE_MAGNETIC_FIELD:
            //取得磁場感應器的數值
            magnitude_values = (float[]) event.values.clone();
            break;
```

```
        default:
            break;
    }
    //其他程式碼
}
```

呼叫 SensorManager.getRotationMatrix() 並依據加速度感應器的數值來計算旋轉矩陣（rotation matrix）。

```
/* 用來儲存下面 accelerometer_values 參數（第 3 個參數）計算出來的旋轉矩陣 */
float[] R = new float[9];
/* 第 2 個參數設定爲 null 是因爲不需要地磁傾斜度的資訊，但第 4 個參數 magnitude_values
不可爲 null，否則會產生 Exception */
SensorManager.getRotationMatrix(R, null, accelerometer_values,
magnitude_values);
```

呼叫 SensorManager.getOrientation() 並依據旋轉矩陣計算出行動裝置的方位。

```
float[] values = new float[3]; //儲存由 R 計算出來的方位資訊
SensorManager.getOrientation(R, values);
```

要取得方位資訊所需相關方法，說明如表 10-3：

▼表 10-3

SensorManager 類別
public static boolean getRotationMatrix (float[] R, float[] I, float[] gravity, float[] geomagnetic) 將參數 gravity 數值轉成對應的旋轉矩陣並儲存在 R 參數內；將參數 geomagnetic 數值轉成對應的地磁傾斜矩陣儲存在 I 參數內。換句話說，就是依據加速度感應器的數值來計算旋轉矩陣；依據磁場感應器的數值來計算地磁傾斜矩陣。如果成功回傳 true，失敗回傳 false（例如自由落體）。 • R：float 陣列，內含 9 個 float 數字，用來儲存旋轉矩陣。 • I：float 陣列，內含 9 個 float 數字，用來儲存地磁傾斜矩陣。 • gravity：float 陣列，內含 3 個 float 數字，必須指定加速度感應器（TYPE_ACCELEROMETER）的數值。 • geomagnetic：float 陣列，內含 3 個 float 數字，必須指定磁場感應器（TYPE_MAGNETIC_FIELD）的數值。

SensorManager 類別
public static float[] getOrientation (float[] R, float[] values) 依據旋轉矩陣計算出行動裝置的方位。旋轉矩陣的座標系統屬於世界座標系統（the world coordinate system）與行動裝置的方位座標系統不同，所以必須加以轉換。回傳值與 values 參數的值相同。 • R：旋轉矩陣，其數值來自於 getRotationMatrix (float[] R, float[] I, float[] gravity, float[] geomagnetic)第 1 個參數 R。 • values：行動裝置的方位數值，是一個內含 3 個 float 數字的陣列，用來儲存 R 計算完畢的結果。

呼叫 getOrientation() 會回傳 float 陣列，如前所述，通常以 values[i] 來代表各種旋轉情形，單位是**弧度**（radians），其意義說明如下：

- values[0]：方位角（azimuth），行動裝置以羅盤方式旋轉（沿著 Z 軸旋轉），會改變方位角的值。如果符合圖 10-11 的 azimuth 箭頭方向旋轉，值會變大（0 → π）[4]；反向則會變小（0 → -π）。
- values[1]：投擲角（pitch），行動裝置以投擲方式旋轉（沿著 X 軸旋轉），會改變投擲角的值。如果符合圖 10-11 的 pitch 箭頭方向旋轉，值會變大（0 → π）；反向則會變小（0 → -π）。
- values[2]：滾動角（roll），行動裝置以滾動方式旋轉（沿著 Y 軸旋轉），會改變滾動角的值。如果符合圖 10-11 的 roll 箭頭方向旋轉，值會變大（0 → π）；反向則會變小（0 → -π）。

[4] values 內的值都是弧度（radians），而非角度（degrees）；0 → π 其實就是角度的變化是由 0 度→180 度，而 π 是圓週率，值近似於 3.14159。

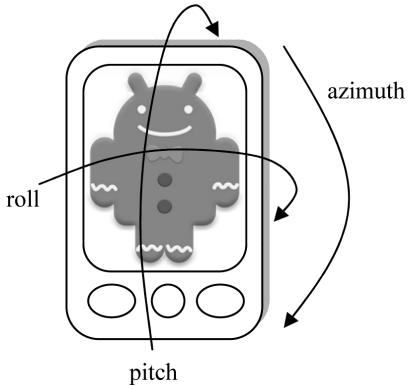

圖 10-11

範例 OrientationEx

圖 10-12

範例說明：

- 依據加速度感應器的數值計算出行動裝置目前的方位數值並顯示在畫面上。
- values[0]、values[1]、values[2] 分別代表行動裝置方位角、投擲角、滾動角的弧度；值都介於 -3.14159 ~ +3.14159 之間。

OrientationEx/src/org/orientationEx/OrientationEx.java

```
30.    SensorEventListener listener = new SensorEventListener() {
31.        public void onSensorChanged(SensorEvent event) {
32.            switch (event.sensor.getType()) {
33.                case Sensor.TYPE_ACCELEROMETER:
34.                    accelerometer_values = (float[]) event.values.clone();
35.                    break;
36.                case Sensor.TYPE_MAGNETIC_FIELD:
37.                    magnitude_values = (float[]) event.values.clone();
38.                    break;
39.                default:
40.                    break;
41.            }
42.
43.            if (magnitude_values != null && accelerometer_values != null) {
44.                float[] R = new float[9];
45.                float[] values = new float[3];
46.                SensorManager.getRotationMatrix(R, null,
47.                        accelerometer_values, magnitude_values);
48.                SensorManager.getOrientation(R, values);
49.                StringBuilder sensorInfo = new StringBuilder();
50.                for (int i = 0; i < values.length; i++)
51.                    sensorInfo.append("-values[" + i + "] = " + values[i] + "\n");
52.                tvMsg.setText(sensorInfo);
53.            }
54.        }
55.
56.        public void onAccuracyChanged(Sensor sensor, int accuracy) {}
57.    };
58.
59.    @Override
60.    protected void onResume() {
61.        super.onResume();
62.        if(!(sensorMgr.registerListener(listener, sensorMgr
63.                .getDefaultSensor(Sensor.TYPE_ACCELEROMETER),
64.                SensorManager.SENSOR_DELAY_UI) &&
65.            sensorMgr.registerListener(listener, sensorMgr
66.                .getDefaultSensor(Sensor.TYPE_MAGNETIC_FIELD),
67.                SensorManager.SENSOR_DELAY_UI))){
68.            Log.w("OrientationEx", "sensor not found!");
69.            sensorMgr.unregisterListener(listener);
70.        }
71.    }
72.
```

```
73.     @Override
74.     protected void onPause() {
75.         super.onPause();
76.         sensorMgr.unregisterListener(listener);
77.     }
```

32-41 行：取得感應器的種類後，判斷如果屬於加速度感應器，就將感應器數值存入 accelerometer_values 變數；如果屬於磁場感應器，就將感應器數值存入 magnitude_values 變數。因為 event.values 取得的陣列內容會隨時變動，所以必須呼叫 clone() 將陣列的值複製一份而非僅是傳址。

43 行：當 accelerometer_values 與 magnitude_values 不為 null 時；換句話說，就是取得加速度感應器與磁場感應器的數值時。

46-47 行：呼叫 getRotationMatrix()，依據加速度感應器的數值來計算旋轉矩陣，並將結果存入 R 變數。第 2 個參數設定為 null 是因為不需要地磁傾斜度的資訊。

48-52 行：呼叫 getOrientation()，依據 R 變數計算出行動裝置的方位數值，並將結果存入 values 變數。之後將 values 內的數值一一顯示在 TextView 元件上。

62-70 行：如果加速度感應器與磁場感應器有任何一個無法運作，以 log 檔記錄並解除感應器的註冊。

10-3-2 透過「Sensor.TYPE_ORIENTATION」取得方位資訊

使用此方法取得方位感應器數值的方式與加速度感應器完全相同，所以不再贅述。values 陣列內的 3 個元素值的單位是**角度**（degrees）而非弧度，所代表的意義說明如下：