

# 第 23 章

## 传感器应用

现在的 Android 手机中，都会内置一些传感器。通过这些传感器可以监测手机上发生的物理事件，而我们只要灵活的运用这些事件，就可以开发出很多方便、实用的 APP。本章将对 Android 中的传感器进行详细介绍。

### 23.1 Android 传感器概述

传感器是一种微型的物理设备，能够探测、感受到外界信号，并按一定规律转换成我们需要的信息。在 Android 系统中，提供了用于接收这些信息并传递给我们的 API。利用这些 API 就可以开发出想要的功能。

Android 系统中的传感器可用于监视设备的移动和位置以及周围环境的变化。例如，实现类似微信摇一摇的功能时，如图 23.1 所示，可以使用加速度传感器来监听各个方向的加速度值；实现类似神庙逃亡 2 游戏时，如图 23.2 所示，可以使用方向传感器来实现倾斜设备变道功能。

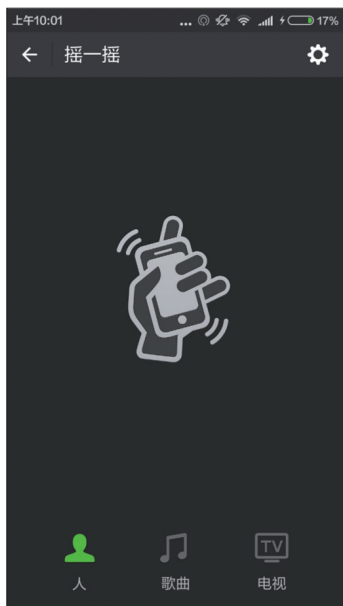


图 23.1 微信摇一摇



图 23.2 神庙逃亡 2

23.1.1 Android 的常用传感器

目前市场上很多 App 都使用到传感器。比如在一些 App 中可以自动识别屏幕的横屏或竖屏方向来改变屏幕布局，这是因为手机硬件支持重力感应和方向判断等功能。实际上 Android 系统对所有类型的传感器的处理都是一样的，只是传感器的类型有所区别。

与传感器硬件进行交互需要使用 Sensor 对象。Sensor 对象描述了它们代表的硬件传感器的属性，其中包括传感器的类型、名称、制造商以及与精确度和范围有关的详细信息。


Sensor 类包含了一组常量，这些常量描述了一个特定的 Sensor 对象所表示的硬件传感器的类型。形式为 Sensor.TYPE\_<TYPE>。在 Android 中支持的传感器的类型如表 23.1 所示。

表 23.1 Android 中支持的传感器类型

名 称	传感器类型常量	描 述
加速度传感器	Sensor.TYPE_ACCELEROMETER	用于获取 Android 设备在 X、Y、Z 三个方向上的加速度，单位为 m/s <sup>2</sup>
重力传感器	Sensor.TYPE_GRAVITY	返回一个三维向量，这个三维向量可显示重力的方向和强度，单位为 m/s <sup>2</sup> 。其坐标系统与加速度传感器的坐标系统相同
线性加速度传感器	Sensor.TYPE_LINEAR_ACCELEROMETER	用于获取 Android 设备在 X、Y、Z 三个方向上不包括重力的加速度，单位为 m/s <sup>2</sup> 。加速度传感器、重力传感器和线性加速度传感器这三者输出值的计算公式如下：加速度传感器 = 重力传感器 + 线性加速度传感器
陀螺仪传感器	Sensor.TYPE_GYROSCOPE	用于获取 Android 设备在 X、Y、Z 这三个方向上的旋转速度，单位是弧度 / 秒。该值为正值时代表逆时针旋转，该值为负值时代表顺时针旋转
光线传感器	Sensor.TYPE_LIGHT	用于获取 Android 设备所处外界环境的光线强度，单位是勒克斯（Lux 简称 lx）
磁场传感器	Sensor.TYPE_MAGNETIC_FIELD	用于获取 Android 设备在 X、Y、Z 三个方向上的磁场数据，单位是微特斯拉（μT）
方向传感器	Sensor.TYPE_ORIENTATION	返回三个角度，这三个角度可以确定设备的摆放状态
压力传感器	Sensor.TYPE_PRESSURE	用于获取 Android 设备所处环境的压力的大小，单位为毫巴（millibars）
距离传感器	Sensor.TYPE_PROXIMITY	用于检测物体与 Android 设备的距离，单位是厘米。一些距离传感器只能返回“远”和“近”两个状态，“远”表示传感器的最大工作范围，而“近”是指比该范围小的任何值

续表

名 称	传感器类型常量	描 述
温度传感器	Sensor.TYPE_AMBIENT_TEMPERATURE	用于获取 Android 设备所处环境的温度，单位是摄氏度。这个传感器是在 Android 4.0 中引入的，用于代替已被弃用的 Sensor.TYPE_TEMPERATURE
相对湿度传感器	Sensor.TYPE_RELATIVE_HUMIDITY	用于获取 Android 设备所处环境的相对湿度，以百分比的形式表示。这个传感器是在 Android 4.0 中引入的
旋转矢量传感器	Sensor.TYPE_ROTATION_VECTOR	返回设备的方向，它表示为 X、Y、Z 三个轴的角度组合，是一个将坐标轴和角度混合计算得到的数据

 **说明** 虽然Android系统中支持多种传感器类型，但并不是每个Android设备都完全支持这些传感器。

### 23.1.2 开发步骤

开发传感器应用大致需要经过以下 3 个步骤：

（1）调用 Context 的 getSystemService(Context.SENSOR\_SERVICE) 方法来获取 SensorManager 对象。SensorManager 是所有传感器的一个综合管理类，包括传感器的种类、采样率、精准度等。调用 Context 的 getSystemService() 方法的代码如下：

```
SensorManager sensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
```

（2）调用 SensorManager 的 getDefaultSensor(int type) 方法来获取指定类型的传感器。例如，返回默认的压力传感器的代码如下：

```
Sensor defaultPressure = sensorManager.getDefaultSensor(Sensor.TYPE_PRESSURE);
```

（3）在 Activity 的 onResume() 方法中调用 SensorManager 的 registerListener() 方法为指定传感器注册监听器。程序通过实现监听器即可获取传感器传回来的数据。调用 registerListener() 方法的语法格式如下：

```
sensorManager.registerListener(SensorEventListener listener, Sensor sensor, int rate)
```

参数说明如下：

- ◆ listener：监听传感器事件的监听器。该监听器需要实现 SensorEventListener 接口。
- ◆ sensor：传感器对象。
- ◆ rate：指定获取传感器数据的频率，它支持的频率值如表 23.2 所示。

表 23.2 获取传感器数据的频率值

频 率 值	描 述
SensorManager.SENSOR_DELAY_FASTEST	尽可能快地获得传感器数据，延迟最小
SensorManager.SENSOR_DELAY_GAME	适合游戏的频率

续表

频 率 值	描 述
SensorManager.SENSOR_DELAY_NORMAL	正常频率
SensorManager.SENSOR_DELAY_UI	适合普通用户界面的频率，延迟较大

例如，使用正常频率为默认的压力传感器注册监听器的代码如下：

```
sensorManager.registerListener(this, defaultPressure, SensorManager.SENSOR_DELAY_NORMAL);
```

◆ SensorEventListener 是使用传感器的核心，其中需要实现的两个方法如下：

◎ onSensorChanged(SensorEvent event) 方法

该方法在传感器的值发生改变时调用。其参数是一个 SensorEvent 对象，通过该对象的 values 属性可以获取传感器的值，该值是一个包含了已检测到的新值的浮点型数组。不同传感器所返回的值的个数及其含义是不同的。不同传感器的返回值的详细信息如表 23.3 所示。

表 23.3 传感器的返回值

传感器名称	值的数量	值的构成	注 释
重力传感器	3	value[0]: X 轴 value[1]: Y 轴 value[2]: Z 轴	沿着三个坐标轴以 $\text{m/s}^2$ 为单位的重力
加速度传感器	3	value[0]: X 轴 value[1]: Y 轴 value[2]: Z 轴	沿着三个坐标轴以 $\text{m/s}^2$ 为单位的加速度
线性加速度传感器	3	value[0]: X 轴 value[1]: Y 轴 value[2]: Z 轴	沿着三个坐标轴以 $\text{m/s}^2$ 为单位的加速度，不包含重力
陀螺仪传感器	3	value[0]: X 轴 value[1]: Y 轴 value[2]: Z 轴	绕三个坐标轴的旋转速率，单位是弧度 / 秒
光线传感器	1	value[0]: 照度	以勒克斯 (Lux) 为单位测量的外界光线强度
磁场传感器	3	value[0]: X 轴 value[1]: Y 轴 value[2]: Z 轴	以微特斯拉为单位表示的环境磁场
方向传感器	3	value[0]: X 轴 value[1]: Y 轴 value[2]: Z 轴	以角度确定设备的摆放状态
压力传感器	1	value[0]: 气压	以毫巴为单位测量的气压
距离传感器	1	value[0]: 距离	以厘米为单位测量的设备与目标的距离

续表

传感器名称	值的数量	值的构成	注 释
温度传感器	1	value[0]: 温度	以摄氏度为单位测量的环境温度
相对湿度传感器	1	value[0]: 相对湿度	以百分比形式表示的相对湿度
旋转矢量传感器	3 (还有一个可选参数)	value[0]: $x*\sin(\theta/2)$ value[1]: $y*\sin(\theta/2)$ value[2]: $z*\sin(\theta/2)$ value[3]: $\cos(\theta/2)$ (可选)	设备方向, 以绕坐标轴的旋转角度表示

传感器的坐标系统和 Android 设备屏幕的坐标系统不同。对于大多数传感器来说, 其坐标系统的 X 轴方向沿屏幕向右, Y 轴方向沿屏幕向上, Z 轴方向是垂直屏幕向上。传感器的坐标系统如图 23.3 所示。

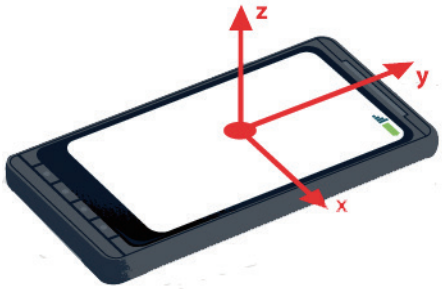


图 23.3 传感器的坐标系统

**注意** 在Android设备屏幕的方向发生改变时, 传感器坐标系统的各坐标轴不会发生变化, 即传感器的坐标系统不会因设备的移动而改变。

◎ onAccuracyChanged(Sensor sensor, int accuracy) 方法

该方法在传感器的精度发生改变时调用。参数 sensor 表示传感器对象, 参数 accuracy 表示该传感器新的精度值。

以上就是开发传感器的 3 个步骤。除此之外, 当应用程序不再需要接收更新时, 需要注销其传感器事件监听器, 代码如下:

```
sensorManager.unregisterListener(this);
```

**说明** Android模拟器本身并没有提供传感器的功能, 开发者需要把程序部署到具有传感器的物理设备上运行。

下面通过一个实例来演示传感器的开发步骤。

例 23.1 实时输出重力传感器和光线传感器的值

在 Android Studio 中创建一个 Module, 名称为 “Sensor Test”。实现本实例的具体步骤如下:

(1) 修改新建 Module 的 res/layout 目录下的布局文件 activity\_main.xml, 首先将默认添加的布局管理器修改为垂直的线性布局管理器, 然后在布局管理器中添加用于显示传感器名称的文本框组件与用于显示传感器输出信息的编辑框组件。

(2) 打开默认添加的 MainActivity, 然后实现 SensorEventListener 接口, 再重写相应的方法, 并定义所需的成员变量, 最后在 onCreate() 方法中, 获取布局管理器中添加的编辑框组件, 并获取传感器管理对象, 具体代码如下:

```

01 public class MainActivity extends AppCompatActivity implements SensorEventListener {
02     EditText textGRAVITY, textLIGHT;           //传感器输出信息的编辑框
03     private SensorManager sensorManager;       //定义传感器管理器
04     @Override
05     protected void onCreate(Bundle savedInstanceState) {
06         super.onCreate(savedInstanceState);
07         setContentView(R.layout.activity_main);
08         //获取重力传感器输出信息的编辑框
09         textGRAVITY= (EditText) findViewById(R.id.textGRAVITY);
10         //获取光线传感器输出信息的编辑框
11         textLIGHT= (EditText) findViewById(R.id.textLIGHT);
12         //获取传感器管理
13         sensorManager= (SensorManager) getSystemService(SENSOR_SERVICE);
14     }
15     @Override
16     public void onSensorChanged(SensorEvent event) {
17     }
18     @Override
19     public void onAccuracyChanged(Sensor sensor, int accuracy) {
20     }
21 }

```

(3) 重写 onResume() 方法, 实现当界面获取焦点时为传感器注册监听器, 具体代码如下:

```

01 @Override
02 protected void onResume() {
03     super.onResume();
04     //为重力传感器注册监听器
05     sensorManager.registerListener(this,
06         sensorManager.getDefaultSensor(Sensor.TYPE_GRAVITY),
07         SensorManager.SENSOR_DELAY_GAME);
08     //为光线传感器注册监听器
09     sensorManager.registerListener(this,
10         sensorManager.getDefaultSensor(Sensor.TYPE_LIGHT),
11         SensorManager.SENSOR_DELAY_GAME);
12 }

```

(4) 重写 onPause() 与 onStop() 方法, 并且在这两个方法中取消注册的监听器, 具体代码如下:

```

01 @Override
02 protected void onPause() { //取消注册监听器
03     sensorManager.unregisterListener(this);

```

```

04     super.onPause();
05 }
06 @Override07     protected void onStop() { //取消注册监听器
08     sensorManager.unregisterListener(this);
09     super.onStop();
10 }

```

(5) 重写 onSensorChanged() 方法, 在该方法中首先获取传感器 X、Y、Z 三个轴的输出信息, 然后获取传感器类型, 并输出相应传感器的信息, 关键代码如下:

```

01 float[] values = event.values; //获取X、Y、Z三轴的输出信息
02 int sensorType = event.sensor.getType(); //获取传感器类型
03 switch (sensorType) {
04     case Sensor.TYPE_GRAVITY:
05         StringBuilder stringBuilder = new StringBuilder();
06         stringBuilder.append("X轴横向重力值:");
07         stringBuilder.append(values[0]);
08         stringBuilder.append("\nY轴纵向重力值:");
09         stringBuilder.append(values[1]);
10         stringBuilder.append("\nZ轴向上重力值:");
11         stringBuilder.append(values[2]);
12         textGRAVITY.setText(stringBuilder.toString());
13         break;
14     case Sensor.TYPE_LIGHT:
15         stringBuilder = new StringBuilder();
16         stringBuilder.append("光的强度值:");
17         stringBuilder.append(values[0]);
18         textLIGHT.setText(stringBuilder.toString());
19         break;
20 }

```

(6) 在 AndroidManifest.xml 文件的 <activity> 标记中添加 screenOrientation 属性, 设置其竖屏显示, 关键代码如下:

```
android:screenOrientation="portrait"
```

(7) 运行本实例, 将显示如图 23.4 所示。



图 23.4 获取传感器输出信息



## 23.2 方向传感器

方向传感器简称为 O-sensor，它用于感应 Android 设备的摆放状态。方向传感器可以返回三个角度，第 1 个代表在 Z 轴上旋转的角度、第 2 个代表在 X 轴上旋转的角度、第 3 个代表在 Y 轴上旋转的角度。

在以前的 Android SDK 中，我们可以通过 SensorManager 对象的 getDefaultSensor(Sensor.TYPE\_ORIENTATION) 方法可以获取到方向传感器，但是在最新版本的 SDK 中提示这种方式已过期，不建议使用，因此 Google 建议使用加速度传感器和磁场传感器组合计算出方向和角度值。其步骤如下：

- (1) 获得加速度传感器和磁场传感器的实例，并为它们注册监听器。
- (2) 在 onSensorChanged() 方法中，分别获取加速度传感器和磁场传感器的值，并传入到 getRotationMatrix() 方法中，从而得出一个包含旋转矩阵的 R 数组。该数组中保存着磁场和加速度的数据。getRotationMatrix() 方法的语法如下：

```
public static boolean getRotationMatrix
    (float[] R, float[] I, float[] gravity, float[] geomagnetic)
```

getRotationMatrix() 方法的参数如表 23.4 所示。

表 23.4 getRotationMatrix() 方法的参数

参 数	描 述
R	需要填充的 float 型数组，大小是 9
I	一个转换矩阵，将磁场数据转换进实际的重力坐标中，一般情况下可以设置为 null
gravity	一个大小为 3 的 float 型数组，表示从加速度传感器获取来的数据
geomagnetic	一个大小为 3float 型的数组，表示从磁场传感器获取来的数据

(3) 通过 SensorManager.getOrientation() 方法来获得所需的旋转数据。getOrientation() 方法的语法如下：

```
public static float[] getOrientation (float[] R, float[] values)
```

- ☑ 参数 R 是步骤 (2) 得到的旋转矩阵，通过该值求出方位角；
- ☑ 参数 values 是一个包括 3 个元素的 float 类型的数组，手机在各个方向上的旋转数据都会被保存到这个数组中。每个数组元素代表的值如表 23.5 所示。

表 23.5 values 数组的数组元素描述

数 组 元 素	描 述
values[0]	手机在 Z 轴上旋转时，手机顶部朝向与正北方的夹角。如果用“磁场 + 加速度”的方式得到的数据范围是 -180 ~ 180 度，也就是说，0 度表示正北，90 度表示正东，180/-180 度表示正南，-90 度表示正西；而直接通过方向传感器得到的数据范围是 0 ~ 359 度，0 度表示正北，90 度表示正东，180 度表示正南，270 度表示正西



续表

数 组 元 素	描 述
values[1]	手机在 X 轴上旋转时（即手机前后翻转时）手机与水平面形成的夹角，手机顶部向上抬起时，该角度的范围是 0~90 度，手机尾部向上抬起时，该角度的范围是 0~90 度
values[2]	手机在 Y 轴上旋转时（即手机左右翻转时）手机与水平面形成的夹角，手机左侧抬起时，该角度的范围是 0~90 度，手机右侧抬起时，该角度的范围是 0~90 度

表 18.5 中的 values[0]、values[1] 和 values[2] 代表的旋转方向如图 23.5 所示。

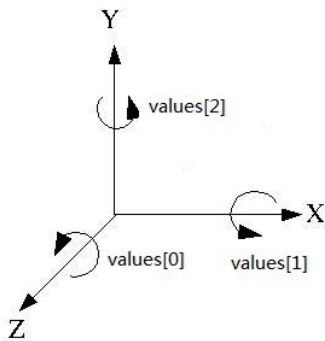


图 23.5 values 数组各元素代表的旋转方向

**注意** 通过getOrientation()方法计算得到的数据是以弧度为单位的，一般情况下，我们都会使用角度为旋转角度的单位，所以需要使用Math.toDegrees()方法进行转换。例如，将values[0]转换为角度可以使用下面的代码：

```
Math.toDegrees(values[0]);
```

通过使用方向传感器，应用程序就可以检测到设备的摆放状态，比如手机顶部或尾部的朝向、倾斜角度等。因此，借助于方向传感器可以开发出水平仪等应用。

下面通过一个实例来演示方向传感器的应用。

例 23.2

通过方向传感器实现一个水平仪

在 Android Studio 中创建 Module，名称为“Level”，实现本实例的具体步骤如下：

（1）创建一个名称为 SpiritlevelView 的类，该类继承自 android.view.View 类并且实现 SensorEventListener 接口，再重写相应的方法。具体代码如下：

```
01 public class SpiritlevelView extends View implements SensorEventListener {
02     public SpiritlevelView(Context context, @Nullable AttributeSet attrs) {
03         super(context, attrs);
04     }
05     @Override
06     public void onSensorChanged(SensorEvent event) {
07     }
08     @Override
09     public void onAccuracyChanged(Sensor sensor, int accuracy) {
10     }
11 }
```

(2) 修改布局文件 activity\_main.xml, 首先将默认添加的布局管理器修改为帧布局管理器, 然后将内边距与默认添加的 TextView 组件删除并设置背景图片。最后在帧布局管理器中添加步骤 (1) 中创建的自定义 View。

(3) 打开 SpiritlevelView 类, 在 SpiritlevelView 类中, 定义所需的成员变量。关键代码如下:

```
01 private Bitmap bubble;           //定义水平仪中的小蓝球位图
02 private int MAX_ANGLE = 30;      //定义水平仪最大倾斜角, 超过该角度, 小蓝球将直接位于边界
03 private int bubbleX, bubbleY;    //定义水平仪中小蓝球的X、Y坐标
```

(4) 在 SpiritlevelView 类的构造方法中, 首先获取要绘制的小蓝球位图与传感器管理, 然后为磁场传感器和加速度传感器注册监听器。具体代码如下:

```
01 public SpiritlevelView(Context context, AttributeSet attrs) {
02     super(context, attrs);
03     bubble = BitmapFactory           //加载小蓝球图片
04         .decodeResource(getResources(), R.drawable.bubble);
05     SensorManager sensorManager = (SensorManager) context
06         .getSystemService(Context.SENSOR_SERVICE); //获取传感器管理
07     sensorManager.registerListener(this,           //为磁场传感器注册监听器
08         sensorManager.getDefaultSensor(Sensor.TYPE_MAGNETIC_FIELD),
09         SensorManager.SENSOR_DELAY_GAME);
10     sensorManager.registerListener(this,           //为加速度传感器注册监听器
11         sensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER),
12         SensorManager.SENSOR_DELAY_GAME);
13 }
```

(5) 在 SpiritlevelView 类中, 创建传感器的取值数组, 关键代码如下:

```
01 float[] accelerometerValues = new float[3]; //创建加速度传感器Z轴、X轴、Y轴取值数组
02 float[] magneticValues = new float[3];      //创建磁场传感器Z轴、X轴、Y轴取值数组
```

(6) 重写 onSensorChanged() 方法, 在该方法中首先获取方向信息, 然后调用 getPosition() 方法计算小篮球的动态位置。关键代码如下:

```
01 @Override
02 public void onSensorChanged(SensorEvent event) {
03     //如果当前为加速度传感器
04     if (event.sensor.getType() == Sensor.TYPE_ACCELEROMETER) {
05         //将取出的值放到加速度传感器取值数组中
06         accelerometerValues = event.values.clone();
07         //如果当前为磁场传感器
08     } else if (event.sensor.getType() == Sensor.TYPE_MAGNETIC_FIELD) {
09         magneticValues = event.values.clone(); //将取出的值放到磁场传感器取值数组中
10     }
11     float[] R = new float[9]; //创建存放旋转数据的取值数组
12     float[] values = new float[3]; //创建存放方向数据的取值数组
13     SensorManager.getRotationMatrix(R, null, accelerometerValues, magneticValues);
14     SensorManager.getOrientation(R, values); //获取方向Z轴、X轴、Y轴信息值
```

```

15     float xAngle = (float) Math.toDegrees(values[1]);    //获取与X轴的夹角
16     float yAngle = (float) Math.toDegrees(values[2]);    //获取与Y轴的夹角
17     getPosition(xAngle,yAngle);                          //获取小蓝球的位置坐标
18     super.postInvalidate();                              //刷新界面
19 }

```

(7) 编写自定义方法 `getPosition()`，用于根据 X 轴和 Y 轴的旋转角度确定小蓝球的位置，具体代码如下：

```

01. private void getPosition(float xAngle,float yAngle){
02.     //小蓝球位于中间时（水平仪完全水平），小蓝球的X、Y坐标
03.     int x = (super.getWidth() - bubble.getWidth()) / 2;
04.     int y = (super.getHeight() - bubble.getHeight()) / 2;
05.     /*****控制小球的X轴位置*****/
06.     if (Math.abs(yAngle) <= MAX_ANGLE) {    //如果Y轴的倾斜角度还在最大角度之内
07.         //根据Y轴的倾斜角度计算X坐标的变化值（倾斜角度越大，X坐标变化越大）
08.         int deltaX = (int)
09.             ((super.getWidth() - bubble.getWidth()) / 2 * yAngle / MAX_ANGLE);
10.         x -= deltaX;
11.         //如果Y轴的倾斜角度已经大于MAX_ANGLE，小蓝球在最左边
12.     } else if (yAngle > MAX_ANGLE) {
13.         x = 0;
14.     } else { //如果与Y轴的倾斜角已经小于负的MAX_ANGLE，小蓝球在最右边
15.         x = super.getWidth() - bubble.getWidth();
16.     }
17.     /*****控制小球的Y轴位置*****/
18.     if (Math.abs(xAngle) <= MAX_ANGLE) {    //如果X轴的倾斜角度还在最大角度之内
19.         //根据X轴的倾斜角度计算Y坐标的变化值（倾斜角度越大，Y坐标变化越大）
20.         int deltaY = (int)
21.             ((super.getHeight() - bubble.getHeight()) / 2 * xAngle / MAX_ANGLE);
22.         y += deltaY;
23.         //如果与X轴的倾斜角度已经大于MAX_ANGLE，小蓝球在最下边
24.     } else if (xAngle > MAX_ANGLE) {
25.         y = super.getHeight() - bubble.getHeight();
26.     } else { //如果X轴的倾斜角已经小于负的MAX_ANGLE，小蓝球在最上边
27.         y = 0;
28.     }
29.     //更新小蓝球的坐标
30.     bubbleX = x;
31.     bubbleY = y;
32. }

```

(8) 重写 `onDraw()` 方法，在该方法中首先根据方向传感器的坐标绘制小蓝球的位置。关键代码如下：

```

01. @Override
02. protected void onDraw(Canvas canvas) {
03.     super.onDraw(canvas);

```

```

04    //根据小蓝球坐标绘制小蓝球
05    canvas.drawBitmap(bubble, bubbleX, bubbleY, null);
06 }

```

(9) 在 AndroidManifest.xml 文件的 <activity> 标记中添加 screenOrientation 属性, 设置其竖屏显示, 关键代码如下:

```
android:screenOrientation="portrait"
```

(10) 运行本实例, 将显示如图 23.6 所示。

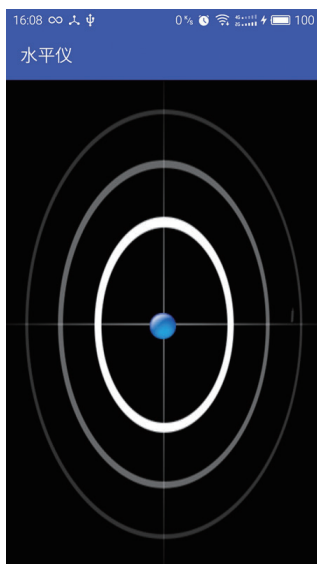


图 23.6 方向传感器的水平仪

## 23.3 磁场传感器

磁场传感器简称为 M-sensor, 主要用于读取 Android 设备外部的磁场强度。随着 Android 设备位置移动和摆放状态的改变, 周围的磁场在设备 X、Y、Z 三个坐标方向上的影响也会发生改变。

磁场传感器会返回 3 个数据, 这 3 个数据分别代表 X、Y、Z 三个坐标方向上的磁场数据。该数值的单位是微特斯拉 ( $\mu\text{T}$ )。

通过使用磁场传感器, 应用程序就可以检测到设备周围的磁场强度, 因此, 借助于磁场传感器可以开发出指南针等应用。

下面通过一个实例来演示磁场传感器的应用。

### 例 23.3 使用磁场传感器实现指南针

在 Android Studio 中创建 Module, 名称为 “Compass”。实现本实例的具体步骤如下:

(1) 创建一个名称为 PointerView 类, 该类继承自 android.view.View 类并且实现 SensorEventListener 接口, 再重写相应的方法, 最后定义所需的成员变量, 具体代码如下:

```

01 public class PointerView extends View implements SensorEventListener {
02     private Bitmap pointer = null;           //定义指针位图
03     private float[] allValue;                //定义传感器三轴的输出信息
04     private SensorManager sensorManager;     //定义传感器管理器
05     public PointerView(Context context, AttributeSet attrs) {
06         super(context, attrs);
07     }
08     @Override
09     public void onSensorChanged(SensorEvent event) {
10     }
11     @Override
12     public void onAccuracyChanged(Sensor sensor, int accuracy) {
13     }
14     @Override
15     protected void onDraw(Canvas canvas) {
16         super.onDraw(canvas);
17     }
18 }

```

(2) 修改布局文件 activity\_main.xml，首先将默认添加的布局管理器修改为帧布局管理器，然后将默认添加的 TextView 组件删除。并且在帧布局管理器中添加一个 ImageView 组件，用于显示背景图，最后添加步骤 (1) 中创建的自定义 View。修改后的代码如下：

```

01 <?xml version="1.0" encoding="utf-8"?>
02 <FrameLayout
03     xmlns:android="http://schemas.android.com/apk/res/android"
04     xmlns:tools="http://schemas.android.com/tools"
05     android:layout_width="match_parent"
06     android:layout_height="match_parent"
07     tools:context="com.mingrisoft.MainActivity">
08     <ImageView
09         android:id="@+id/background"
10         android:layout_width="wrap_content"
11         android:layout_height="wrap_content"
12         android:layout_gravity="center"
13         android:src="@drawable/background"
14     />
15     <!--添加自定义View-->
16     <com.mingrisoft.PointerView
17         android:layout_width="wrap_content"
18         android:layout_height="wrap_content" />
19 </FrameLayout>

```

(3) 打开 PointerView 类，在 PointerView 类的构造方法中，首先获取要绘制的指针位图与传感器管理器，然后为磁场传感器注册监听器。关键代码如下：

```

01 pointer = BitmapFactory.decodeResource(super.getResources(),

```

```

02      R.drawable.pointer); //获取要绘制的指针位图
03  //获取传感器管理器
04  sensorManager = (SensorManager) context
05      .getSystemService(Context.SENSOR_SERVICE);
06  //为磁场传感器注册监听器
07  sensorManager.registerListener(this,
08      sensorManager.getDefaultSensor(Sensor.TYPE_MAGNETIC_FIELD),
09      SensorManager.SENSOR_DELAY_GAME);

```

(4) 重写 onSensorChanged() 方法, 在该方法中首先判断获取到的是否是磁场传感器, 然后获取磁场传感器 X、Y、Z 三个坐标轴的输出信息并保存信息, 最后通过 super.postInvalidate() 方法刷新界面。关键代码如下:

```

01  if (event.sensor.getType() == Sensor.TYPE_MAGNETIC_FIELD) { //如果是磁场传感器
02      float value[] = event.values; //获取磁场传感器三轴的输出信息
03      allValue = value; //保存输出信息
04      super.postInvalidate(); //刷新界面
05  }

```

(5) 重写 onDraw() 方法, 在该方法中首先根据磁场传感器的坐标计算指针的角度, 然后绘制指针。关键代码如下:

```

01  Paint p = new Paint(); //创建画笔
02  if (allValue != null) { //传感器三轴输出信息不为空
03      float x = allValue[0]; //获取X轴坐标
04      float y = allValue[1]; //获取Y轴坐标
05      canvas.save(); //保存Canvas的状态
06      canvas.restore(); //重置绘图对象
07      //以屏幕中心点作为旋转中心
08      canvas.translate(super.getWidth() / 2, super.getHeight() / 2);
09      //判断Y轴为0时的旋转角度
10      if (y == 0 && x > 0) {
11          canvas.rotate(90); //旋转角度为90度
12      } else if (y == 0 && x < 0) {
13          canvas.rotate(270); //旋转角度为270度
14      } else {
15          //通过三角函数tanh()方法计算旋转角度
16          if (y >= 0) {
17              canvas.rotate((float) Math.tanh(x / y) * 90);
18          } else {
19              canvas.rotate(180 + (float) Math.tanh(x / y) * 90);
20          }
21      }
22  }
23  //绘制指针
24  canvas.drawBitmap(this.pointer, -this.pointer.getWidth() / 2,
25      -this.pointer.getHeight() / 2, p);

```



(6) 在 AndroidManifest.xml 文件的 <activity> 标记中添加 screenOrientation 属性，设置其竖屏显示，关键代码如下：

```
android:screenOrientation="portrait"
```

(7) 运行本实例，将显示如图 23.7 所示的界面效果。

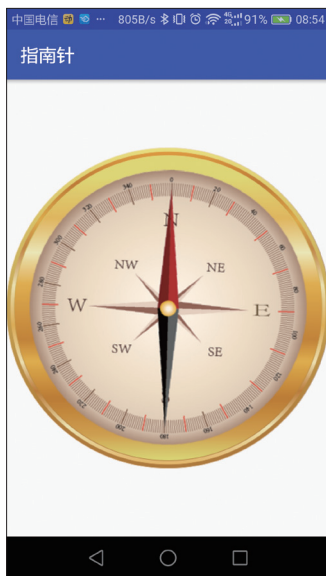


图 23.7 指南针

## 23.4 加速度传感器

加速度传感器是用于检测设备加速度的传感器。对于加速度传感器来说，SensorEvent 对象的 values 属性将返回 3 个值，分别代表 Android 设备在 X、Y、Z 三个坐标方向上的加速度，单位为  $\text{m/s}^2$ 。当 Android 设备横向左右移动时，可能产生 X 轴上的加速度，当 Android 设备前后移动时，可能产生 Y 轴上的加速度，当 Android 设备垂直上下移动时，可能产生 Z 轴上的加速度。

通过使用加速度传感器，可以开发出类似微信摇一摇以及运动 App 的计步功能。

下面通过一个实例来演示加速度传感器的应用。

### 例 23.4 使用加速度传感器实现摇红包

在 Android Studio 中创建 Module，名称为“Shake Red Packet”。实现本实例的具体步骤如下：

(1) 修改布局文件 activity\_main.xml，首先将默认添加的布局管理器修改为相对布局管理器，然后将 TextView 组件删除，再为布局管理器添加背景。

(2) 创建一个名称为 packet.xml 的布局文件，在该布局文件中添加一个 ImageView 组件，用于显示红包图片。

(3) 打开默认添加的 MainActivity，让 MainActivity 实现 SensorEventListener 接口，再重写相应的方法，最后在该类中，定义所需的成员变量，关键代码如下：

```

01 private SensorManager sensorManager;    //定义传感器管理器
02 private Vibrator vibrator;              //定义振动器

```

(4) 在 onCreate() 方法中, 获取传感器管理器与振动器服务。关键代码如下:

```

01 sensorManager = (SensorManager) getSystemService(SENSOR_SERVICE); //获取传感器管理器
02 vibrator = (Vibrator) getSystemService(Service.VIBRATOR_SERVICE); //获取振动器服务

```

(5) 重写 onResume() 方法, 并在该方法中为传感器注册监听器。具体代码如下:

```

01 @Override
02 protected void onResume() {
03     super.onResume();
04     //为加速度传感器注册监听器
05     sensorManager.registerListener(this, sensorManager.getDefaultSensor
06         (Sensor.TYPE_ACCELEROMETER), SensorManager.SENSOR_DELAY_GAME);
07 }

```

(6) 重写 onSensorChanged() 方法, 实现摇动手机, 显示红包的功能。具体代码如下:

```

01 @Override
02 public void onSensorChanged(SensorEvent event) {
03     float[] values = event.values;           //获取传感器X、Y、Z三个坐标轴的输出信息
04     int sensorType = event.sensor.getType(); //获取传感器类型
05     if (sensorType == Sensor.TYPE_ACCELEROMETER) { //如果是加速度传感器
06         //X轴输出信息>15,Y轴输出信息>15,Z轴输出信息>20
07         if (values[0] > 15 || values[1] > 15 || values[2] > 20) {
08             Toast.makeText(MainActivity.this, "摇一摇", Toast.LENGTH_SHORT).show();
09             //创建AlertDialog.Builder对象
10             AlertDialog.Builder alertDialog = new AlertDialog.Builder(this);
11             alertDialog.setView(R.layout.packet); //添加布局文件
12             alertDialog.show(); //显示alertDialog
13             vibrator.vibrate(500); //设置振动器频率
14             sensorManager.unregisterListener(this); //取消注册的监听器
15         }
16     }
17 }

```

(7) 在 AndroidManifest.xml 文件的 <activity> 标记中添加 screenOrientation 属性, 设置其竖屏显示, 关键代码如下:

```

android:screenOrientation="portrait"

```

(8) 打开 AndroidManifest.xml 文件, 在其中设置振动器的使用权限, 具体代码如下:

```

<uses-permission android:name="android.permission.VIBRATE"></uses-permission>

```

(9) 运行本实例, 将显示如图 23.8 所示的界面, 摇动手机后显示如图 23.9 所示的红包。