Android 手机特性

Android 提供了对设备传感器的支持，只要 Android 设备的硬件提供了这些传感器，Android 应用可以通过传感器来获取设备的外界条件，包括手机的运行状态、当前摆放的方向等。Android 系统还提供了驱动程序去管理这些传感器硬件，可以通过监听器的方式监听传感器硬件感知到的外部环境的变化。

开发传感器应用步骤

开发一个对于传感器支持的应用十分简单，开发人员只要在传感器管理器 SensorManager 中为所要监听的传感器指定一个监听器即可，当外部环境发生变化的时候，Android 系统会通过传感器获取外部环境的数据，然后将数据传递给监听器的监听回调方法。具体步骤如下：

1. 获取传感器服务。
2. 从传感器服务中获取到指定类型的传感器。
3. 使用传感器服务添加传感器的监听器。
4. 在使用完之后，注销传感器的监听器。

获取传感器服务

Android 中内置了很多系统级的服务，用于给开发人员使用，而传感器也是通过传感器服务，SensorManager 来管理的。而在 Android 组件中获取系统服务，使用方法 Context.getSystemService(String)即可，它的参数均以 static final 的方式定义在 Context 中，而获取 SensorManager 需要传入 Context.SENSOR_SERVICE。

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

从传感器服务中获取到指定类型的传感器

传感器服务管理设备上所有的传感器，所以需要指定待监听的传感器。获取待监听的传感器，需要使用 SensorManager.getDefaultSensor()方法，它的完整签名如下：

Sensor getDefaultSensor(int type)

Android 中的传感器需要 Sensor 支持，getDefaultSensor()方法通过指定的 type 参数获取到相对应的传感器。type 参数被以 static final 的方式定义在 Sensor 内部，方便开发人员可以直接使用。下面是几个常用传感器的 type：

- Sensor.TYPE_ORIENTATION：方向传感器。
- Sensor.TYPE_ACCELEROMETER：重力传感器。
- Sensor.TYPE_LIGHT：光线传感器。
- Sensor.TYPE_MAGNETIC_FIELD：磁场传感器。

使用传感器服务添加传感器的监听器

获得 SensorManager 和 Sensor 对象之后，就可以为其 Sensor 注册监听器了。为传感器注册监听器，使用 SensorManager.registerListener()方法即可，它存在多个重载方法，但是有些方法已经过时了，下面提供一个常用的方法的签名：

boolean registerListener(SensorEventListener listener,Sensor sensor,int rateUs)

上面方法参数的意义：listener：传感器的监听器、sensor：待监听的传感器、rateUs：传感器的采样率。

从 registerListener()方法可以看出，它需要传递一个 SensorEventListener 对象，它就是传

感器的监听器，其中包含两个方法，需要开发人员去实现它：

- void onAccuracyChanged(Sensor sensor,int accuracy)：当传感器精度发生变化时回调。
- void onSensorChanged(SensorEvent event)：当传感器感应的值发生变化时回调。

对于上面两个方法，传感器的精度一般是不会发生改变的，所以我们一般主要的代码量在 onSensorChanged()中。

registerListener()方法还有一个 rateUs 的参数，它表示监听传感器改变的采样率，就是从传感器获取值的频率。它被定义以 static final 的形式定义在 SensorManager 中，方便我们直接使用，它定义了如下几个选项：

- SensorManager.SENSOR_DELAY_FASTEST：最快，延迟最小。
- SensorManager.SENSOR_DELAY_GAME：适合游戏的频率。
- SensorManager.SENSOR_DELAY_NORMAL：正常频率。
- SensorManager.SENSOR_DELAY_UI：适合普通用户界面 UI 变化的频率。

Android 为我们提供了这几个采样率的参数，方便我们使用。但对于选择那种采样率而言，并不是越快越好，要参照实际开发的应用的情况来说，采样率越大，将越耗费资源，包括电量、CPU 等，所以要根据实际情况选择，毕竟再强大的应用，如果造成设备续航能力的降低，也是会被用户所不喜的。

在使用完之后，注销传感器的监听器

当使用完传感器之后，需要为其注销监听器，因为传感器的监听器并不会因为应用的结束而自行释放资源，需要开发人员在适当的时候主动注销。注销传感器监听器使用 SensorManager.unregisterListener()方法即可，和监听器的注册方法一样，它也具有多个重载的方法，但是有一些已经被弃用了，下面介绍一个常用的完整签名：

void unregisterListener(SensorEventListener listener)

指南针 Demo

上面已经讲解了在应用中使用传感器的步骤以及具体内容，下面通过一个简单的 Demo 来演示一下如何使用传感器。在 Demo 中监听方向传感器，使其角度的变化改变来操作方向，模拟一个指南针的效果。

重写监听器的 onSensorChanged()方法，其中 event 获取当当前监听事件的参数，可以使用 values[0]获取到当前的方向传感器感应到的角度。参照官方文档，可以看出，它代表一个 360°的角度，规则是： 0=North, 90=East, 180=South, 270=West。

需要注意的是传感器的 Demo 需要在真机上测试，因为模拟器上不存在传感器硬件。

Step1:

把指南针图片 compass.png 放在 MySensorManager\app\src\main\res\drawable 目录下。

Step2:

在 app\res\layout\acitvity_main.xml 中设计 UI，添加一个 ImageView。

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/activity_main"
    android:layout_width="match_parent"
```

```
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context="com.example.mysensormanager.MainActivity">

    <ImageView android:id="@+id/iv_compass"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:layout_centerVertical="true"
        android:src="@drawable/compass" />
</RelativeLayout>
```

Step3:

在 app/java/com.examle.mysensormanager/MainActivity 下添加实现代码。

```
public class MainActivity extends AppCompatActivity {
    private ImageView iv_compass;
    private SensorManager manager;
    private float startDegree = 0f;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        iv_compass = (ImageView) findViewById(R.id.iv_compass);
        // 获得传感器管理器
        manager = (SensorManager) getSystemService(SENSOR_SERVICE);
    }

    @Override
    protected void onResume() {
        super.onResume();
        // 为方向传感器注册监听器
        manager.registerListener(listener,
                manager.getDefaultSensor(Sensor.TYPE_ORIENTATION),
                SensorManager.SENSOR_DELAY_UI);
    }

    private SensorEventListener listener = new SensorEventListener() {
        @Override
        public void onSensorChanged(SensorEvent event) {
            if (event.sensor.getType() == Sensor.TYPE_ORIENTATION) {
```

```
                // 获取当前传感器获取到的角度
                float degree = -event.values[0];
                // 通过补间动画旋转角度，从上次的角度开始旋转
                RotateAnimation ra = new RotateAnimation(startDegree,
degree,
                        Animation.RELATIVE_TO_SELF, 0.5f,
                        Animation.RELATIVE_TO_SELF, 0.5f);
                ra.setDuration(200);
                iv_compass.startAnimation(ra);
                // 记录当前旋转后的角度
                startDegree = degree;
            }
        }

        @Override
        public void onAccuracyChanged(Sensor sensor, int accuracy) {

        }
    };

    @Override
    protected void onStop() {
        // 为传感器注销监听器
        manager.unregisterListener(listener);
        super.onStop();
    }
}
```

Step4:

真机测试。