#12 特色开发（传感器）

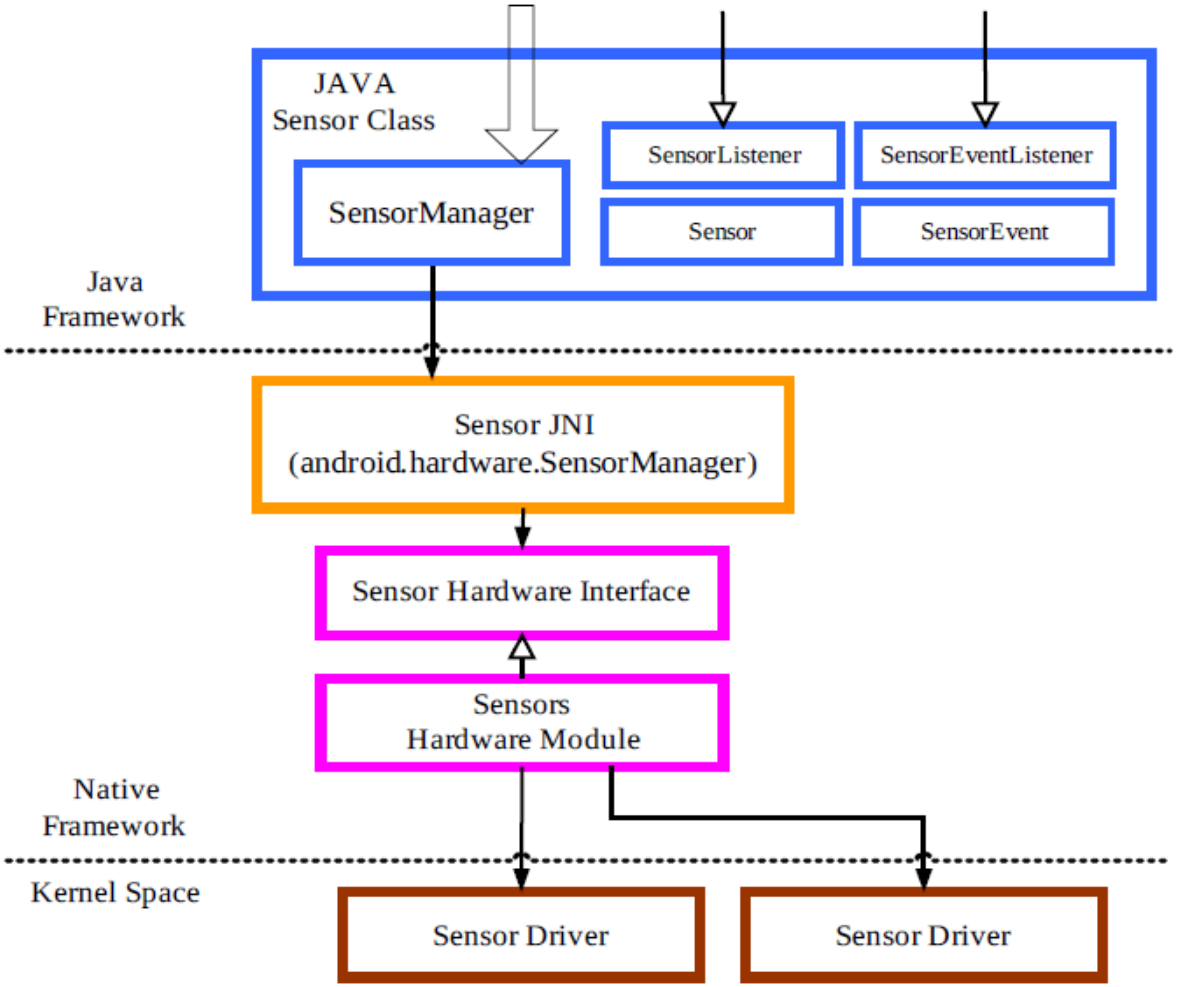
学习目标

- 传感器系统概述
- 传感器系统层次结构
- 传感器系统的硬件抽象层
- 常见的传感器
- 传感器系统的使用

Android 传感器系统概述

- 传感器系统可以让智能手机的功能更加丰富多彩
- Android的Sensor系统涉及了Android的各个层次。
- Android系统支持多种传感器，有的传感器已经在Android的框架中使用，大多数传感器由应用程序来使用。

Sensor系统层次结构



Sensor系统层次结构

Android的传感器系统从驱动程序层次到上层都有所涉及，自下而上涉及到的各个层次为：

- 各种 Sensor内核中的驱动程序
- Sensor的硬件抽象层（硬件模块）
- Sensor系统的JNI（Java Native Interface）
- Sensor的JAVA类
- JAVA框架中对Sensor的使用
- JAVA应用程序对Sensor的使用

Sensor系统层次结构

Sensor模块的初始化函数: sensors_module_init()

```
static jint
sensors_module_init(JNIEnv *env, jclass clazz)
{
    int err = 0;
    sensors_module_t const* module;
    err = hw_get_module(SENSORS_HARDWARE_MODULE_ID,  // 打开Sensor的硬件模块
                        (const hw_module_t **)&module);
    if (err == 0)
        sSensorModule = (sensors_module_t*)module;
    return err;
}
```

Sensor系统层次结构

Sensor系统的JNI 部分的函数列表 ：

```
static JNINativeMethod gMethods[] = {
    {"nativeClassInit",     "()V", (void*)nativeClassInit },
    {"sensors_module_init","()I", (void*)sensors_module_init },
    {"sensors_module_get_next_sensor", "(Landroid/hardware/Sensor;I)I",
                                (void*)sensors_module_get_next_sensor },
    {"sensors_data_init",   "()I", (void*)sensors_data_init },
    {"sensors_data_uninit","()I", (void*)sensors_data_uninit },
    {"sensors_data_open",   "(Ljava/io/FileDescriptor;)I",
                           (void*)sensors_data_open },
    {"sensors_data_close", "()I", (void*)sensors_data_close },
    {"sensors_data_poll",   "([F[I[J)I", (void*)sensors_data_poll },
};
```

Sensor系统层次结构

通过Android 传感器框架获取传感器及传感器数据，其包含了：

- **SensorManager.java：**

  实现传感器系统核心的管理类SensorManager

- **Sensor.java：**

  单一传感器的描述性文件Sensor

- **SensorEvent.java**：

  表示传感器系统的事件类SensorEvent，提供如下信息：原始传感器数据、传感器类型、数据的准确度、事件的时间戳等。

- **SensorEventListener.java：**

  传感器事件的监听者SensorEventListener接口

- **SensorListener.java：**

  传感器的监听者SensorListener接口（在API Level 3中被弃用）

Sensor系统层次结构

SensorManager 的主要的接口如下所示

```
public class SensorManager extends IRotationWatcher.Stub
{
    public Sensor getDefaultSensor (int type) {// 获得默认的传感器 }
    public List<Sensor> getSensorList (int type) { // 获得传感器列表 }
    public boolean registerListener (SensorEventListener listener,
        Sensor sensor, int rate, Handler handler) { // 注册传感器的监听者 }
    void  unregisterListener(SensorEventListener listener, Sensor sensor)
                                { // 注销传感器的监听者 }
}
```

Sensor系统层次结构

- Sensor 的主要的接口如下所示

```
public class Sensor {
    float  getMaximumRange()  { //  获得传感器最大的范围 }
    String  getName()   { //  获得传感器的名称 }
    float  getPower()   {/ /  获得传感器的耗能 }
    float  getResolution()   { //  获得传感器的解析度 }
    int  getType()   { //  获得传感器的类型 }
    String  getVendor()   { //  获得传感器的 Vendor }
    int  getVersion()   { //  获得传感器的版本 }
}
```

- Sensor类的初始化在SensorManager 的JNI代码中实现，在SensorManager中维护一个Sensor的列表

Sensor系统层次结构

- SensorEvent类比较简单，实际上是Sensor类加上了数值（values），精度（accuracy），时间戳（timestamp）等内容。

- SensorEventListener接口描述了SensorEvent的监听者内容如下所示：

```
public interface SensorEventListener {
    public void onSensorChanged(SensorEvent event);
    public void onAccuracyChanged(Sensor sensor, int accuracy);
}
```

Sensor的硬件抽象层

- hardware/libhardware/include/hardware/目录中的sensors.h是Android传感器系统硬件层的接口。

- Sensor模块的定义如下所示：

```
struct sensors_module_t {
    struct hw_module_t common;
    int (*get_sensors_list)(struct sensors_module_t* module,
                            struct sensor_t const** list);
};
```

Sensor的硬件抽象层

sensors_data_t表示传感器的数据：

```
typedef struct {
    int  sensor;   /* sensor 标识符 */
    union {
        sensors_vec_t  vector;       /* x,y,z 矢量 */
        sensors_vec_t  orientation;  /* 加速度（单位：度） */
        sensors_vec_t  acceleration; /* 加速度（单位： m/s^2） */
        sensors_vec_t  magnetic;     /* 磁矢量（单位： uT） */
        float          temperature;  /* 温度（单位：摄氏度） */
    };
    int64_t        time;            /* 时间（单位： nanosecond） */
    uint32_t       reserved;
} sensors_data_t;
```

Sensor的硬件抽象层

Sensor的控制设备和数据设备

```
struct sensors_control_device_t {
    struct hw_device_t common;
    native_handle_t* (*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);
};
```

```
struct sensors_data_device_t {
    struct hw_device_t common;
    int (*data_open)(struct sensors_data_device_t *dev, native_handle_t* nh);
    int (*data_close)(struct sensors_data_device_t *dev);
    int (*poll)(struct sensors_data_device_t *dev,  sensors_data_t* data);
}
```

Sensor的硬件抽象层

sensor_t表示一个传感器的描述性定义：

```
struct sensor_t {
    const char*     name;         /* 传感器的名称 */
    const char*     vendor;       /* 传感器的 vendor */
    int             version;      /* 传感器的版本 */
    int             handle;       /* 传感器的句柄 */
    int             type;         /* 传感器的类型 */
    float           maxRange;     /* 传感器的最大范围 */
    float           resolution;   /* 传感器的辨析率 */
    float           power;        /* 传感器的耗能（估计值， mA 单位） */
    void*           reserved[9];
}
```

Sensor的硬件抽象层

Sensor的硬件抽象层实现的要点：

- 传感器的硬件抽象层可以支持多个传感器，需要构建一个sensor_t类型的数组。
- 传感器控制设备和数据设备结构，可能被扩展。
- 传感器在Linux内核的驱动程序，很可能使用misc驱动的程序，这时需要在控制设备开发的时候，同样使用open()打开传感器的设备节点。

Sensor的硬件抽象层

- 传感器数据设备poll是实现的重点，需要在传感器没有数据变化的时候实现阻塞，在数据变化的时候返回，根据驱动程序的情况可以使用poll()，read()或者ioctl()等接口来实现。
- sensors_data_t数据结构中的数值，是最终传感器传出的数据，在传感器的硬件抽象层中，需要构建这个数据。

常见的传感器

Android平台支持**三大类传感器**

- **运动传感器**

  运动传感器测量加速力和旋转力，它们包括加速度传感器、重力传感器、陀螺仪、旋转角度传感器。

- **环境传感器**

  环境传感器测量各种周围环境情况，包括环境温度、气压、光强、湿度等。

- **位置传感器**

  位置传感器测量设备的物理位置信息，包括方向传感器和磁力传感器。

常见的传感器

传感器管理器的几个常量

- **传感器类型**

  方向、加速表、光线、磁场、临近性、温度等。

- **采样率**

  最快、游戏、普通、用户界面。当应用程序请求特定的采样率时，其实只是对传感器子系统的一个提示，或者一个建议。不保证特定的采样率可用。

- **准确性**

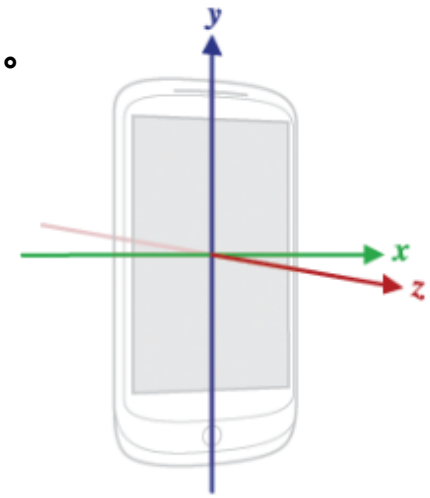
  高、低、中、不可靠。

常见的传感器 世界坐标系

• Android中定义了两个坐标系：**世界坐标系**（world coordinate-system）和**旋转坐标系**（rotation coordinate-system）。

• 世界坐标系定义了从一个特定的Android设备上来看待外部世界的方式，主要是以设备的屏幕为基准而定义。

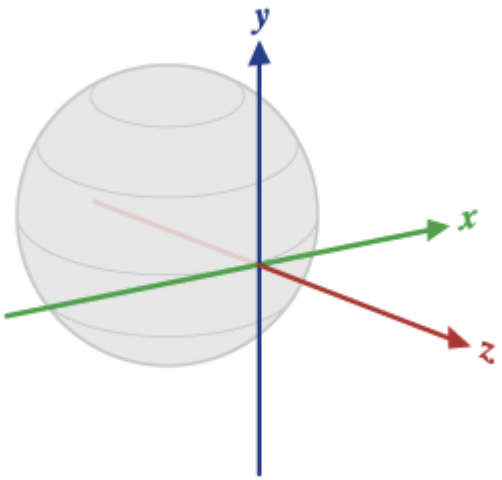
• 坐标系以屏幕的中心为圆点，其中：
  • X轴：方向是沿着屏幕的水平方向从左向右。
  • Y轴：方向与屏幕的侧边平行，方向指向屏幕的顶端。
  • Z轴：将手机屏幕朝上平放在桌面上时，屏幕所朝的方向。



常见的传感器 旋转坐标系

• 旋转坐标系是专用于**方位传感器**（Orientation Sensor）的，方位传感器即用于描述设备所朝向的方向的传感器。方向传感器所传回的数值是屏幕从标准位置（屏幕水平朝上且正北）开始分别以这三个坐标轴为轴所旋转的角度。

- X轴：Y轴与Z轴的向量积Y•Z，与地球球面相切并且指向地理的东方。
- Y轴：为设备当前所在位置与地面相切并且指向地磁北极的方向。
- Z轴：为设备所在位置指向天空的方向，垂直于地面。



常见的传感器

现阶段Android支持的传感器常用有以下几种：

传感器	Android中的名称	描述
加速度传感器	TYPE_ACCELEROMETER	测量在（x, y, z）三个维度上的加速度。单位： m/s^2
环境温度传感器	TYPE_AMBIENT_TEMPERATURE	测量环境的温度。单位：(°C)
重力加速度	TYPE_GRAVITY	测量在（x, y, z）三个维度上的重力加速度。单位： m/s^2
陀螺仪	TYPE_GYROSCOPE	测量在（x, y, z）三个维度上的旋转速度。单位： rad/s
光传感器	TYPE_LIGHT	测量环境的亮度。单位： lx
磁力域	TYPE_MAGNETIC_FIELD	测量在（x, y, z）三个维度上的磁场。单位： μT
方向传感器	TYPE_ORIENTATION	测量手机在（x, y, z）三个维度上的旋转角度。
压力传感器	TYPE_PRESSURE	测量环境的气压。单位：hPa

不同版本的Android系统支持不同的传感器，具体信息可参考:
https://developer.android.com/guide/topics/sensors/sensors_overview.html

常见的传感器 加速度传感器 (Accelerometer)

- 加速度传感器又叫G-sensor，返回x、y、z三轴的加速度(m/s^2)。
- 加速度包含地心引力的影响，使用如下公式计算加速度

$$A_d = -g - \sum F / mass$$

- 将手机平放在桌面上，x轴默认为0，y轴默认0，z轴默认9.81。即设备加速度（0 m/s^2）- 重力加速度（-9.81 m/s^2） = 9.81。

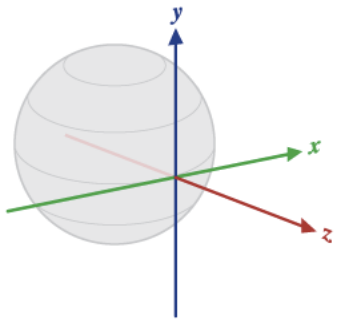
常见的传感器 磁力传感器

- 磁力传感器简称为M-sensor，返回x、y、z三轴的环境磁场数据。
  - 该数值的单位是微特斯拉（micro-Tesla），用uT表示。单位也可以是高斯（Gauss），1Tesla=10000Gauss。
  - 硬件上一般没有独立的磁力传感器，磁力数据由电子罗盘传感器提供（E-compass）。

常见的传感器 方向传感器

- 方向传感器简称为O-sensor，返回三轴的角度数据，方向数据的单位是角度。

- 方向传感器提供三个数据(values)，分别为：侧倾度（azimuth）、俯仰度（pitch）和翻滚度（roll）。
  - **侧倾度**：返回水平时磁北极和Y轴的夹角，范围为0°至360°。
  - **俯仰度**：围绕 x 轴的旋转角。从坐标原点望向x轴正方向，逆时针旋转返回增值，顺时针旋转返回负值。取值范围为-180°到180°。
  - **翻滚度**：围绕 y 轴的旋转角。从y轴正方向望向坐标原点，逆时针旋转返回正值，顺时针返回负值。取值范围为 -90 度到 90 度。



常见的传感器 方向传感器

- 方向传感器直接处理从加速度和磁场传感器中获取的原始数据。这种方法需要大量的计算，因此方向传感器在 Android 2.2 (API level 8)中被废除，方向传感器类型在Android 4.4W (API level 20)中被废除。取而代之使用getRotationMatrix()和getOrientation()方法计算方向数据。

```
// Compute the three orientation angles based on the most recent readings from
// the device's accelerometer and magnetometer.
public void updateOrientationAngles() {
  // Update rotation matrix, which is needed to update orientation angles.
  mSensorManager.getRotationMatrix(mRotationMatrix, null,
    mAccelerometerReading, mMagnetometerReading);

  // "mRotationMatrix" now has up-to-date information.

  mSensorManager.getOrientation(mRotationMatrix, mOrientationAngles);

  // "mOrientationAngles" now has up-to-date information.
}
```

常见的传感器 陀螺仪传感器

- 陀螺仪传感器叫做Gyro-sensor，返回x、y、z三轴的角加速度数据，单位是radians/second。
- 陀螺仪的坐标系与加速度传感器的相同。因此，从 x、y、z 轴的正向位置观看处于原始方位的设备，如果设备逆时针旋转，将会收到正值。
- 手机平放在桌面上，水平顺时针旋转，z轴为负值；逆时针旋转为正值。
- 手机向左旋转，从y轴看为顺时针旋转，因此y轴为负值。

常见的传感器 光线感应传感器

- 光线感应传感器检测实时的光线强度，光强单位是lux，其物理意义是照射到单位面积上的光通量。
- 光线感应传感器主要用于Android系统的LCD自动亮度功能。
- 可以根据采样到的光强数值实时调整LCD的亮度。

常见的传感器 距离传感器

- 大部分距离传感器返回的是**绝对距离**，单位是 cm
- 一些接近传感器只能返回**远和近**两个状态，将最大距离返回远状态，小于最大距离返回近状态。
- 距离传感器通常用于确定用户头部与手持设备屏幕表面的距离，可用于接听电话时自动关闭LCD屏幕以节省电量。
- 一些芯片集成了接近传感器和光线传感器两者功能

常见的传感器

- 压力传感器

压力传感器返回当前的压强，单位是百帕斯卡hectopascal（hPa）。

- 温度传感器

温度传感器返回当前环境的温度。

传感器系统的使用

为了使用手机的传感器功能，需要完成以下步骤：

1. 首先获取传感器服务的引用，这通过创建一个传感器管理器的实例完成

```
// 通过getSystemService获取传感器管理器句柄
sensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
```

2. 从传感器管理器中获取传感器，以陀螺仪传感器为例

```
// 获取默认的陀螺仪传感器
Sensor defaultGyroscope = sensorManager.getDefaultSensor(Sensor.TYPE_GYROSCOPE);
```

传感器系统的使用

3. 注册监听器，监听传感器数据变化

```
// 获取压力传感器
List<Sensor> pressureSensors = sensorManager.getSensorList(Sensor.TYPE_PRESSURE);
```

4. 实现监听器的 SensorEventListner 接口

```
SensorEventListener listener = new SensorEventListener() {
    @Override
    public void onSensorChanged(SensorEvent event) {
        // called when sensor values have changed
    }

    @Override
    public void onAccuracyChanged(Sensor sensor, int accuracy) {
        // called when then accuracy of a sensor has changed
    }
};
```

传感器系统的使用 获取设备上传感器

实际上，从传感器管理器中获取传感器，一共有三种方法

- 第 1 种：获取某种默认的传感器

```
// 获取默认的陀螺仪传感器
Sensor defaultGyroscope = sensorManager.getDefaultSensor(Sensor.TYPE_GYROSCOPE);
```

- 第 2 种：获取某种传感器的列表

```
// 获取压力传感器
List<Sensor> pressureSensors = sensorManager.getSensorList(Sensor.TYPE_PRESSURE);
```

- 第 3 种：获取所有传感器的列表

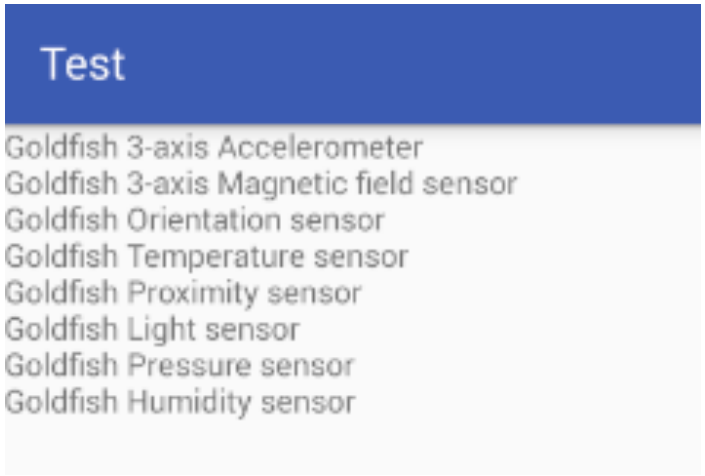
```
// 获取所有传感器
List<Sensor> allSensors = sensorManager.getSensorList(Sensor.TYPE_ALL);
```

传感器系统的使用 获取设备上传感器

```
public class MainActivity extends AppCompatActivity {

    private SensorManager sensorManager;

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

        TextView sensorList = (TextView)findViewById(R.id.sensorlist);

        sensorManager = (SensorManager) getSystemService(SENSOR_SERVICE);

        List<Sensor> sensors = sensorManager.getSensorList(Sensor.TYPE_ALL);

        for(Sensor sensor:sensors)
        {
            //输出传感器的名称
            sensorList.append(sensor.getName() + "\n");
        }
    }
}
```

在模拟器 Nexus 5X API 19 上，通过代码获取到的传感器

传感器系统的使用 获取设备上传感器



在模拟器 Nexus 5X API 19 上，通过代码获取到的传感器

从结果中可以看出，该款手机支持了如下型号的共七种类型的传感器：

- Goldfish 3-axis Accelerometer：三轴加速度传感器
- Goldfish 3-axis Magnetic field sensor：三轴磁场传感器
- Goldfish Orientation sensor：方向传感器
- Goldfish Temperature sensor：温度传感器
- Goldfish Proximity sensor: 近距离传感器
- Goldfish Light sensor：光传感器
- Goldfish Pressure sensor：压力传感器
- Goldfish Humidity sensor：湿度传感器

传感器系统的使用 获取设备上传感器

对于某一个传感器，它的一些具体信息的获取方法

方　法	描　述
getMaximumRange()	最大取值范围
getName()	设备名称
getPower()	功率
getResolution()	精度
getType()	传感器类型
getVentor()	设备供应商
getVersion()	设备版本号

传感器系统的使用 注册和注销传感器监听器

只有在注册了传感器监听器之后，传感器管理器（SensorManager）才会将相应的传感信号传给该监听器。通常将这个注册的操作放在Activity的onResume()方法下，同时将取消注册（即注销）的操作放在Activity的onPause()方法下，这样就可以使传感器的资源得到合理的使用和释放，方法如下：

```
@Override
protected void onResume() {
    super.onResume();
    sensorManager.registerListener(listener, sensorPressure,
            SensorManager.SENSOR_DELAY_UI);
}

@Override
protected void onPause() {
    super.onPause();
    sensorManager.unregisterListener(listener);
}
```

传感器系统的使用 SensorEventListener接口

• Android应用程序中使用传感器要依赖于android.hardware.SensorEventListener接口。通过该接口可以监听传感器的各种事件。

• 接口包括了如上段代码中所声明的两个方法，常用的是onSensorChanged方法，它只有一个SensorEvent类型的参数event。

• SensorEvent类代表了一次传感器的响应事件，当系统从传感器获取到信息的变更时，会捕获该信息并向上层返回一个SensorEvent类型的对象，该对象包含了传感器类型（Sensor sensor）、传感事件的时间戳（long timestamp）、传感器数值的精度（int accuracy）以及传感器的具体数值（float[] values）。

• values值非常重要，其数据类型是float[]，代表了从各种传感器采集回的数值信息。例如，通常温度传感器仅仅传回一个用于表示温度的数值，而加速度传感器则需要传回一个包含X、Y、Z三个轴上的加速度数值。

传感器系统的使用 计步器

• 什么是计步器呢？顾名思义，计步器就是用于计算一个人所走过的步数。那么如何准确的测定步数呢？这就需要借助于传感器了，如何处理、统计传感器的数据，就决定了测定步数的准确性。

• 实现计步器应用需要使用什么传感器？早期实现计步器，都是使用**加速度传感器(Accelerometer Sensor)**测量步数。根据加速度传感器的数据, 绘制空间曲线。根据两次波峰波谷之间的时间间隔, 判断步行或其他状态。同时, 屏蔽轻微与初始扰动, 提升准确性； 通过调整参数, 适配不同手机的传感器差异， 提升鲁棒性.

传感器系统的使用 计步器

- 随后谷歌在android 4.4(KitKat, api 19)推出**TYPE_STEP_DETECTOR** 和 **TYPE_STEP_COUNTER**, 由硬件或系统计算步数的变化, 使得算法简化。

- TYPE_STEP_COUNTER
  记录了从第一次注册以来的所有步数，无论中间unregister 与否，除非是关机reboot ，才会被清零。

- TYPE_STEP_DETECTOR
  每次用户走了一步之后被触发，返回的时间戳是用户的脚触碰地面，产生高速的加速度的时间。detector 具有更高的灵敏性，往往稍微的手表或者手机晃动都可以致使步数增加，

传感器系统的使用 计步器

使用android自带的TYPE_STEP_DETECTOR 和TYPE_STEP_COUNTER传感器实现的计步器

```
// Step Counter
sensorManager.registerListener(new SensorEventListener() {
    @Override
    public void onSensorChanged(SensorEvent event) {
        float steps = event.values[0];
        textViewStepCounter.setText("已经走了 " + (int) steps + " 步");
    }

    @Override
    public void onAccuracyChanged(Sensor sensor, int accuracy) {
    }
}, sensorManager.getDefaultSensor(Sensor.TYPE_STEP_COUNTER),
        SensorManager.SENSOR_DELAY_UI);
```

```
// Step Detector
sensorManager.registerListener(new SensorEventListener() {
    @Override
    public void onSensorChanged(SensorEvent event) {
        step += 1;
        textViewStepDetector.setText("Step Detector: 已经走了 " + (int) step + " 步");
    }

    @Override
    public void onAccuracyChanged(Sensor sensor, int accuracy) {
    }
}, sensorManager.getDefaultSensor(Sensor.TYPE_STEP_DETECTOR),
        SensorManager.SENSOR_DELAY_UI);
```

传感器系统的使用 计步器

传感器系统的使用 位置服务

• 位置服务（Location-Based Services，LBS），又称定位服务或基于位置的服务，融合了GPS定位、移动通信、导航等多种技术，提供了与空间位置相关的综合应用服务

• 位置服务首先在日本得到商业化的应用
2001年7月，DoCoMo发布了第一款具有三角定位功能的手持设备
2001年12月，KDDI发布第一款具有GPS功能的手机

• 基于位置的服务发展迅速，已涉及到商务、医疗、工作和生活的各个方面，为用户提供定位、追踪和敏感区域警告等一系列服务

传感器系统的使用 位置服务

- Android平台支持提供位置服务的API，在开发过程中主要用到LocationManager和LocationProviders对象

- LocationManager可以用来获取当前的位置，追踪设备的移动路线或设定敏感区域，在进入或离开敏感区域时设备会发出特定警报

- LocationProviders是能够提供定位功能的组件集合，集合中的每种组件以不同的技术提供设备的当前位置，区别在于定位的精度、速度和成本等方面

传感器系统的使用 位置服务

使用android的位置服务，需要以下**6个步骤。**

1. 首先需要**获得LocationManager对象。**获取LocationManager可以通过调用android.app.Activity.getSystemService()函数实现。

```
LocationManager locationManager;
String systemService = Context.LOCATION_SERVICE;
locationManager = (LocationManager) this.getSystemService(systemService);
```

- 代码第2行的Context.LOCATION_SERVICE指明获取的服务是位置服务
- 代码第3行的getSystemService()函数，可以根据服务名称获取Android提供的系统级服务

传感器系统的使用 位置服务

Android支持的系统级服务表

Context类的静态常量	值	返回对象	说明
LOCATION_SERVICE	location	LocationManager	控制位置等设备的更新
WINDOW_SERVICE	window	WindowManager	最顶层的窗口管理器
LAYOUT_INFLATER_SERVICE	layout_inflater	LayoutInflater	将XML资源实例化为View
POWER_SERVICE	power	PowerManager	电源管理
ALARM_SERVICE	alarm	AlarmManager	在指定时间接受Intent
NOTIFICATION_SERVICE	notification	NotificationManager	后台事件通知
KEYGUARD_SERVICE	keyguard	KeyguardManager	锁定或解锁键盘
SEARCH_SERVICE	search	SearchManager	访问系统的搜索服务
VIBRATOR_SERVICE	vibrator	Vibrator	访问支持振动的硬件
CONNECTIVITY_SERVICE	connection	ConnectivityManager	网络连接管理
WIFI_SERVICE	wifi	WifiManager	Wi-Fi连接管理
INPUT_METHOD_SERVICE	input_method	InputMethodManager	输入法管理

传感器系统的使用 位置服务

2. 在获取到LocationManager后，还需要**指定LocationManager的定位方法**，然后才能够调用LocationManager。

- LocationManager支持的定位方法有两种
  - GPS定位：可以提供更加精确的位置信息，但定位速度和质量受到卫星数量和环境情况的影响
  - 网络定位：提供的位置信息精度差，但速度较GPS定位快

传感器系统的使用 位置服务

- LocationManager支持定位方法

LocationManager类的静态常量	值	说明
GPS_PROVIDER	gps	使用GPS定位，利用卫星提供精确的位置信息，需要android.permissions.ACCESS_FINE_LOCATION用户权限
NETWORK_PROVIDER	network	使用网络定位，利用基站或Wi-Fi提供近似的位置信息，需要具有如下权限：android.permission.ACCESS_COARSE_LOCATION或android.permission.ACCESS_FINE_LOCATION.

- 以使用GPS定位为例，定义获取位置的方法为GPS定位

```
String provider = LocationManager.GPS_PROVIDER;
```

传感器系统的使用 位置服务

3. **指定位置变化事件频率。**

• LocationManager提供了一种便捷、高效的位置监视方法**requestLocationUpdates()**，可以根据位置的距离变化和时间间隔设定产生位置改变事件的条件，这样可以避免因微小的距离变化而产生大量的位置改变事件。

• LocationManager中设定监听位置变化的代码如下。代码将产生位置改变事件的条件设定为距离改变10米，时间间隔为2秒

```
locationManager.requestLocationUpdates(provider, 2000, 10, locationListener);
```

① 第1个参数是定位的方法，GPS定位或网络定位
② 第2个参数是产生位置改变事件的时间间隔，单位为毫秒
③ 第3个参数是距离条件，单位是米
④ 第4个参数是回调函数，在满足条件后的位置改变事件的处理函数

传感器系统的使用 位置服务

4. 定义处理位置变化事件监听类 **LocationListener**。

```
private LocationListener locationListener = new LocationListener() {
    @Override
    public void onLocationChanged(Location location) {
        // 在设备的位置改变时被调用
    }

    @Override
    public void onStatusChanged(String provider, int status, Bundle extras) {
        // 在提供定位功能的硬件的状态改变时被调用
    }

    @Override
    public void onProviderEnabled(String provider) {
        // 在用户启用具有定位功能的硬件时被调用
    }

    @Override
    public void onProviderDisabled(String provider) {
        // 在用户禁用具有定位功能的硬件时被调用
    }
};
```

传感器系统的使用 位置服务

① onLocationChanged()在设备的位置改变时被调用。

• 函数的参数就是当前改变了的位置，是一个Location对象，其包含了可以确定位置的信息，如经度、纬度和速度等。

• 通过调用Location中的getLatitude()和getLonggitude()方法可以分别获取位置信息中的纬度和经度，示例代码如下

```
double lat = location.getLatitude();
double lng = location.getLongitude();
```

传感器系统的使用 位置服务

5. 通常位置监听器获取第一个位置信息需要很长的时间，这时可以调用getLastKnowLocation()方法**获取一个缓存的位置信息**

```
if (PackageManager.PERMISSION_GRANTED == getPackageManager().checkPermission(permissionName, packageName)) {

    Location location = locationManager.getLastKnownLocation(provider);
```

- 在获取位置之前，需要通过函数**checkPermission**检查程序是否获取到了用户权限。只有获取了权限之后，才可以获取到当前的位置。

传感器系统的使用 位置服务

6. 使用位置服务的应用必须**请求用户位置权限。**

- Android拥有两种位置权限：**ACCESS_COARSE_LOCATION** 和 **ACCESS_FINE_LOCATION**。我们选择的权限决定API返回的位置信息的精度，前者精度较后者低。

- 如果你想使用NETWORK_PROVIDER和GPS_PROVIDER，需要声明ACCESS_FINE_LOCATION权限；如果只是用NETWORK_PROVIDER，需要声明ACCESS_COARSE_LOCATION。

传感器系统的使用 位置服务

- 如果目标设备是**Android5.0 (API 21)及以上**，必须声明程序使用android.hardware.location.network or android.hardware.location.gps 硬件特征。在Android5.0 (API 21)以前，声明ACCESS_COARSE_LOCATION 和ACCESS_FINE_LOCATION权限，就包括了使用两种硬件特征。

- 请求权限需要在AndroidManifest.xml文件中声明，代码如下：

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

<!-- Needed only if your app targets Android 5.0 (API level 21) or higher. -->
<uses-feature android:name="android.hardware.location.gps" />
```

传感器系统的使用 位置服务

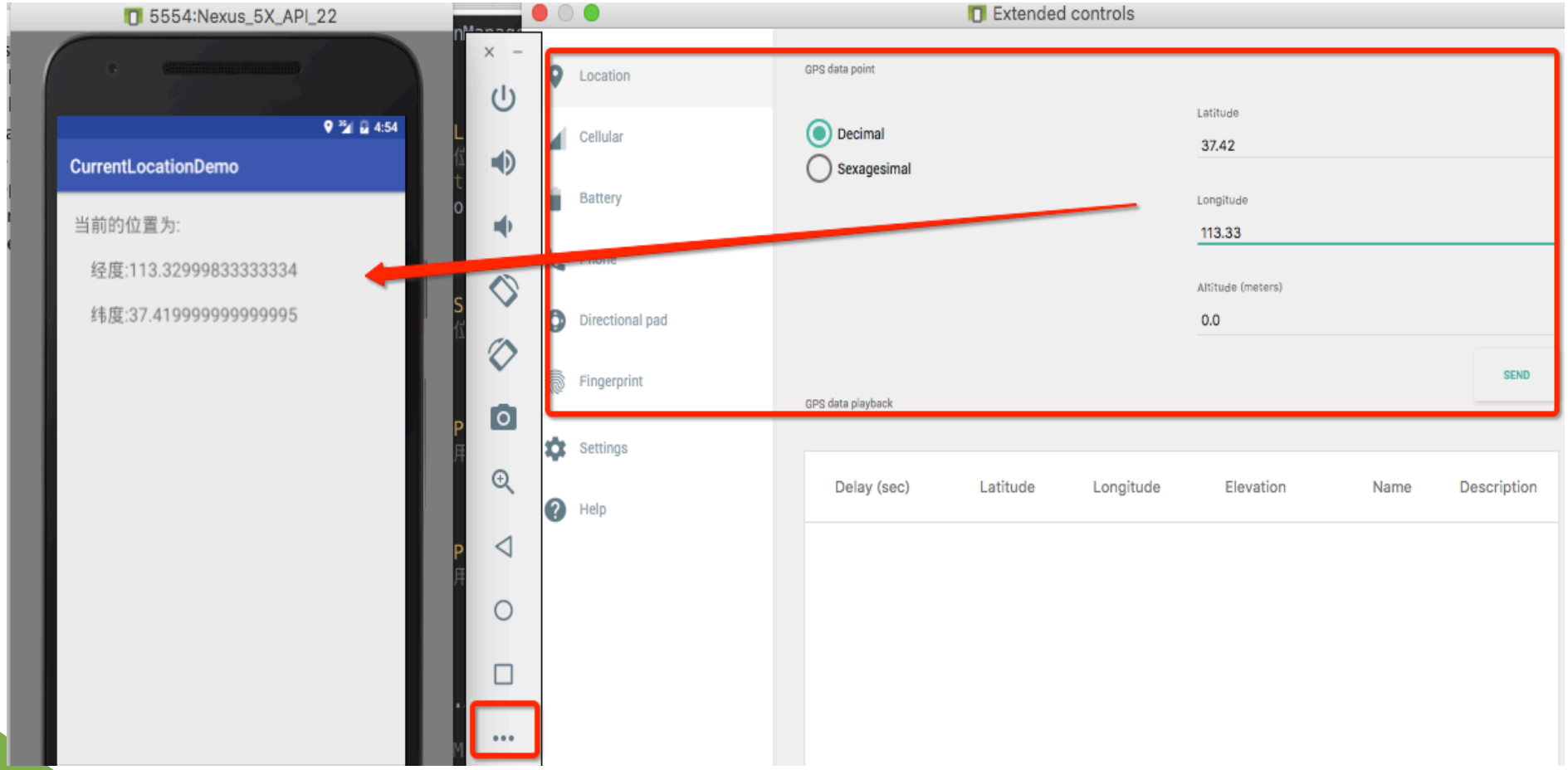
通过以上步骤，实现了一个CurrentLocationDemo的应用，其提供了显示当前位置新的功能，并能够监视设备的位置变化。

传感器系统的使用 位置服务

- 位置服务一般都需要使用设备上的硬件，最理想的调试方式是将程序上传到物理设备上运行，但在没有物理设备的情况下，也可以使用Android模拟器提供的虚拟方式模拟设备的位置变化，调试具有位置服务的应用程序

- 在程序运行过程中，可以在模拟器控制器中改变经度和纬度坐标值，程序在检测到位置的变化后，会将最新的位置信息显示在界面上.

传感器系统的使用 位置服务



传感器系统的使用 位置服务

- 除了使用Android framework location APIs (android.location) 来获取位置之外，还可以使用**Google Play services location API**获取位置信息。

- Google Location Services API 是Google Play Services的一部分，其提供了强大的高级框架来自动处理location provider。Google provider 根据运动和位置精度自动选择provider。

- 此外，Google Location Service 基于电池消耗情况来更新位置信息。

- 使用Google Location Service 能够消耗更少的电量，获取更准确的精度。

Google Play services Location API 使用说明：https://developer.android.com/training/location/index.html

QUESTIONS ?