手机应用平台软件开发

12、特色开发

刘宁
Email：liuning78@mail.sysu.edu.cn

Android的传感器系统

SUN YAT-SEN UNIVERSITY

- 传感器系统综述
- 传感器系统层次结构
- 传感器系统的硬件抽象层
- 传感器系统的使用

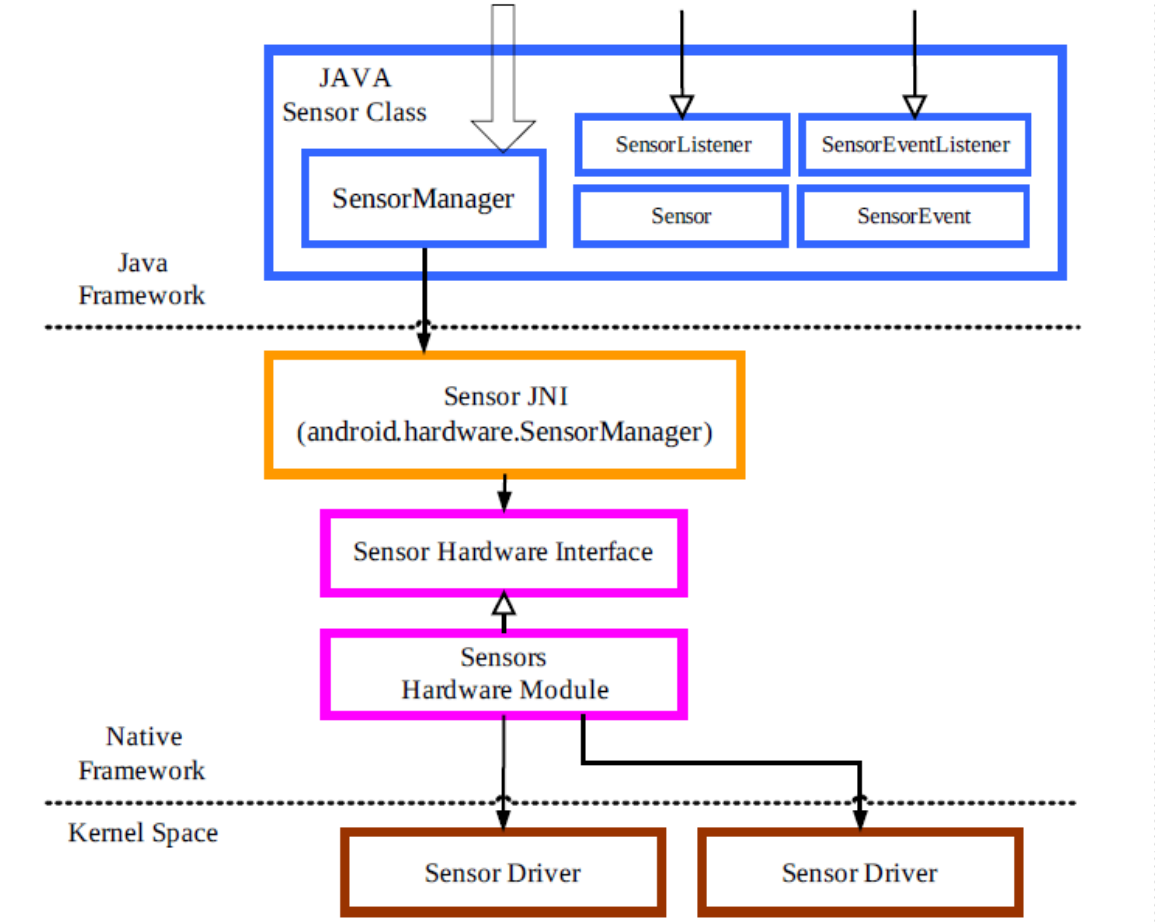
传感器系统综述

- 传感器（Sensor）系统可以让智能手机的功能更加丰富多彩，在Android系统中支持多种传感器。
- 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。

传感器系统综述

传感器	JAVA 中的名称	本地接口名称	数值
加速度	TYPE_ACCELEROMETER	SENSOR_TYPE_ACCELEROMETER	1
磁力域	TYPE_MAGNETIC_FIELD	SENSOR_TYPE_MAGNETIC_FIELD	2
方向	TYPE_ORIENTATION	SENSOR_TYPE_ORIENTATION	3
陀螺	TYPE_GYROSCOPE	SENSOR_TYPE_GYROSCOPE	4
光线（亮度）	TYPE_LIGHT	SENSOR_TYPE_LIGHT	5
压力	TYPE_PRESSURE	SENSOR_TYPE_PRESSURE	6
温度	TYPE_TEMPERATURE	SENSOR_TYPE_TEMPERATURE	7
接近	TYPE_PROXIMITY	SENSOR_TYPE_PROXIMITY	8

传感器系统综述


JAVA
Sensor Class
SensorManager
SensorListener
SensorEventListener
Sensor
SensorEvent
Java
Framework
Sensor JNI
(android.hardware.SensorManager)
Sensor Hardware Interface
Sensors
Hardware Module
Native
Framework
Kernel Space
Sensor Driver
Sensor Driver

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，传感器系统自下而上涉及到的各个层次为：

- 各种Sensor的内核中的驱动程序
- Sensor的硬件抽象层（硬件模块）
- Sensor系统的JNI
- 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

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

SUN YAT-SEN UNIVERSITY

传感器系统的**JAVA**部分包含了以下几个文件

- 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

- 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（不推荐使用）

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 }
}
```

传感器系统综述

- 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.java维护了一个Sensor的列表。

传感器系统层次结构

- 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

SUN YAT-SEN UNIVERSITY

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。
- 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。

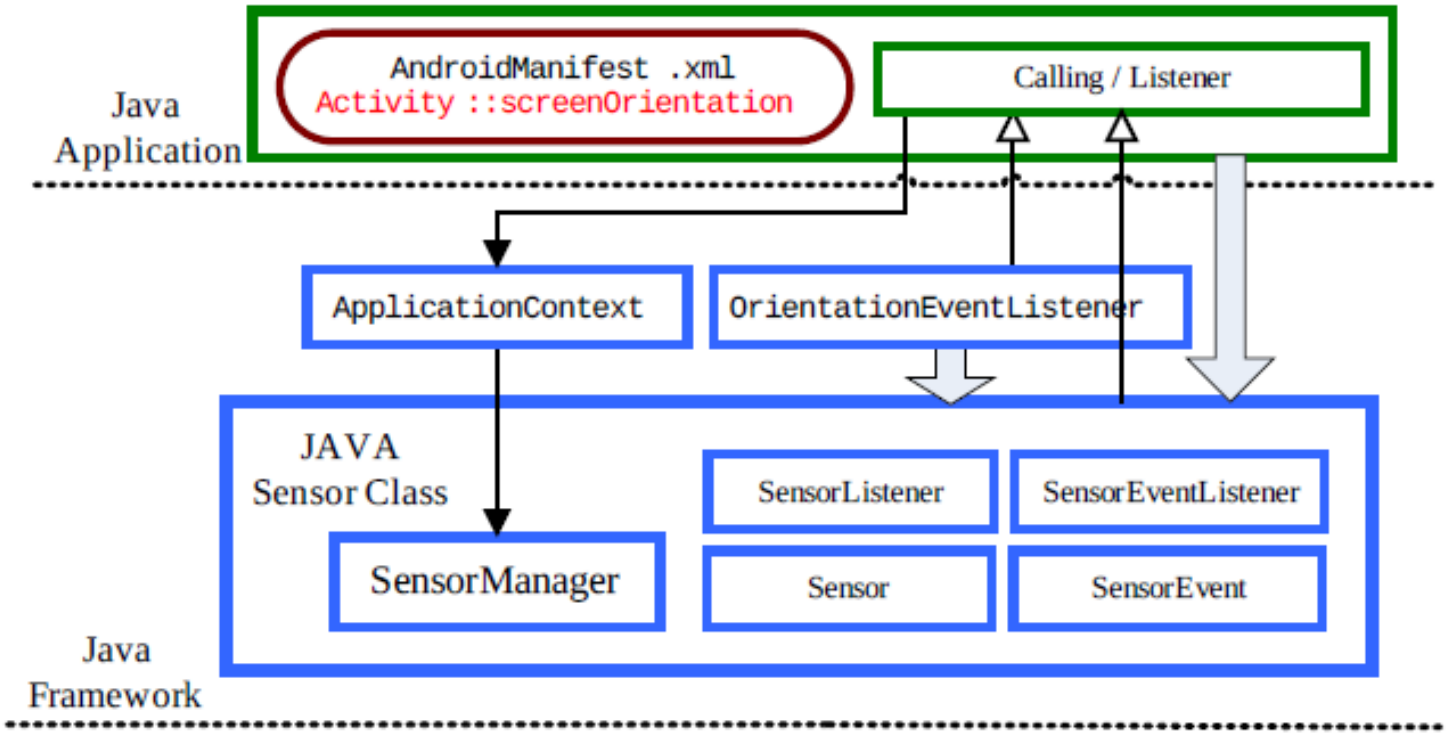
传感器系统的使用

SUN YAT-SEN UNIVERSITY

传感器系统使用的几个方面

- JAVA框架的OrientationEventListener类
- JAVA框架的ApplicationContext
- 应用程序的AndroidManifest.xml设置方向
- 调用传感器系统接口

传感器系统的使用


Java
Application
AndroidManifest .xml
Activity ::screenOrientation
Calling / Listener
ApplicationContext
OrientationEventListener
JAVA
Sensor Class
SensorManager
SensorListener
SensorEventListener
Sensor
SensorEvent
Java
Framework

传感器系统的使用

- Java代码

```
//通过getSystemService获取传感器管理器句柄
mSensorMngr = (SensorManager)mContext.getSystemService(mContext.SENSOR_SERVICE);
//通过getDefaultSensor获取传感器，下面的例子是返回加速度传感器对象
mSensor = mSensorMngr.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
//通过registerListener注册监听器,参数一监听器对象，参数二注册监听的传感器，参数三采样率
mSensorMngr.registerListener(listener, mSensor, SensorManager.SENSOR_DELAY_NORMAL);
```

- 监听器需要实现SensorEventListner接口

```
SensorEventListener listener = new SensorEventListener(){
    @Override
    public void onAccuracyChanged(Sensor sensor, int accuracy) {
        //Called when the accuracy of a sensor has changed.
    }
    @Override
    public void onSensorChanged(SensorEvent event) {
        //Called when sensor values have changed.
    }
};
```

传感器API

SUN YAT-SEN UNIVERSITY

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。SensorEventListener接口如下：

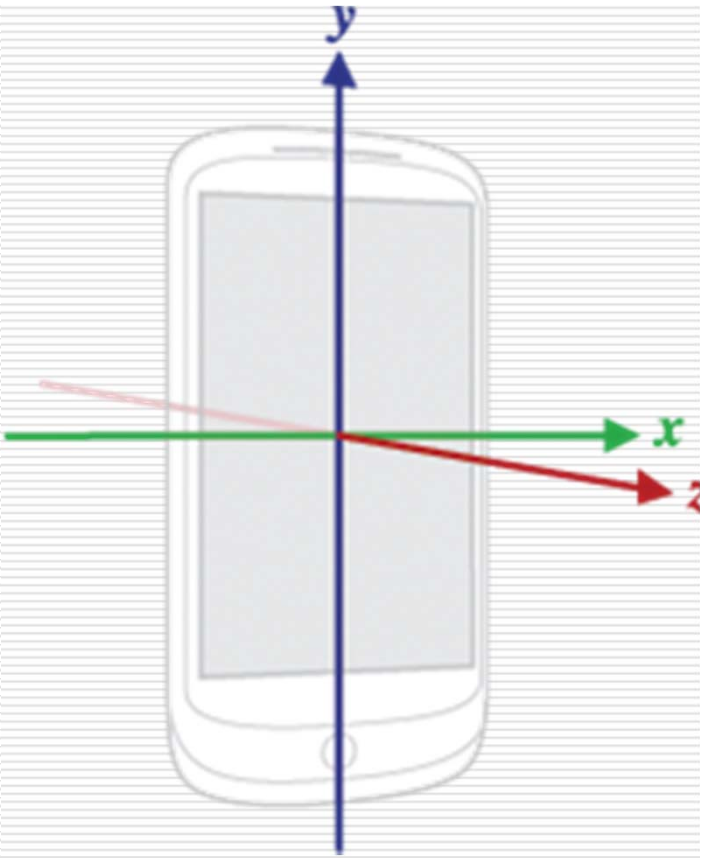
```
package android.hardware;
public interface SensorEventListener {
    //传感器采样值发生变化时调用
    public abstract void onSensorChanged(SensorEvent event);
    //传感器精度发生改变时调用
    public abstract void onAccuracyChanged(Sensor sensor, int accuracy);
}
```

传感器API

- 接口包括了如上段代码中所声明的两个方法，其中onAccuracyChanged方法在一般场合中比较少使用到，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（public Sensor sensor）、传感事件的时间戳（public long timestamp）、传感器数值的精度（public int accuracy）以及传感器的具体数值（public final 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，该float型的数组最多包含3个成员，而根据传感器的不同，values中个成员所代表的含义也不同。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，同样的一个数据“10”，如果是从温度传感器传回则可能代表10摄氏度，而如果从亮度传感器传回则可能代表数值为10的亮度单位，如此等等。

传感器相关的坐标系

- 为了正确理解传感器所传回的数值，先介绍Android所定义的两个坐标系，即世界坐标系（world coordinate-system）和旋转坐标系（rotation coordinate-system）。
- 该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，并且该坐标系依赖的是屏幕的默认方向，不随屏幕显示的方向改变而改变。



世界坐标系

世界坐标系

坐标系以屏幕的中心为圆点，其中：

- X轴

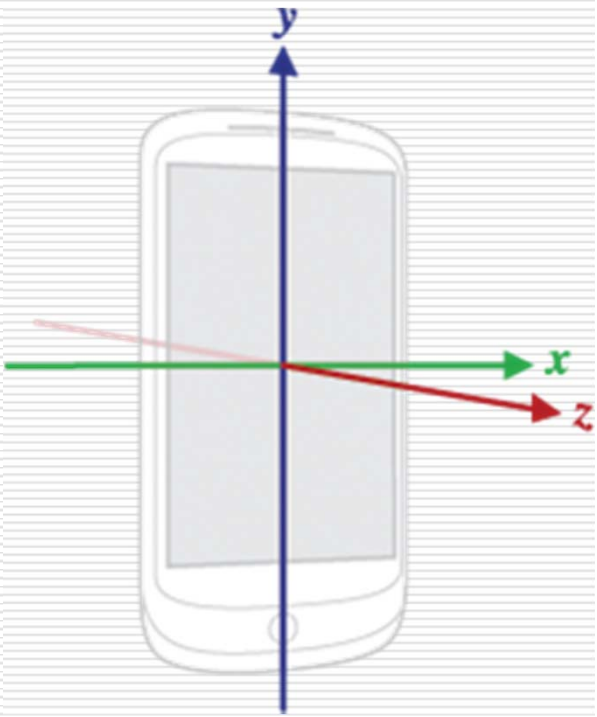
  方向是沿着屏幕的水平方向从左向右。手机默认的正放状态，一般来说即是如图所示的默认为长边在左右两侧并且听筒在上方的情况，如果是特殊的设备，则可能X和Y轴会互换。

- Y轴

  方向与屏幕的侧边平行，是从屏幕的正中心开始沿着平行屏幕侧边的方向指向屏幕的顶端。

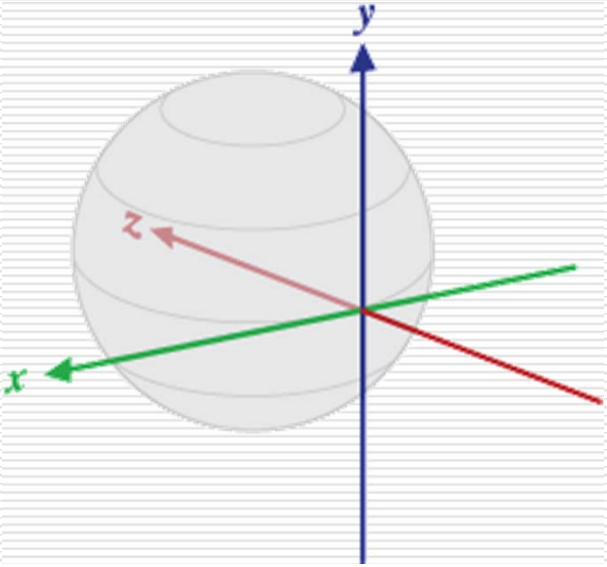
- Z轴

  将手机屏幕朝上平放在桌面上时，屏幕所朝的方向。



旋转坐标系

如图，图中球体可以理解为地球，该坐标系是专用于方位传感器（Orientation Sensor）的，可以理解为一个“反向的（inverted）”世界坐标系，方位传感器即用于描述设备所朝向的方向的传感器，而Android为描述这个方向而定义了一个坐标系，这个坐标系也由X、Y、Z轴构成，特别之处是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。使用方位传感器的典型用例即“电子罗盘”。



旋转坐标系

X轴

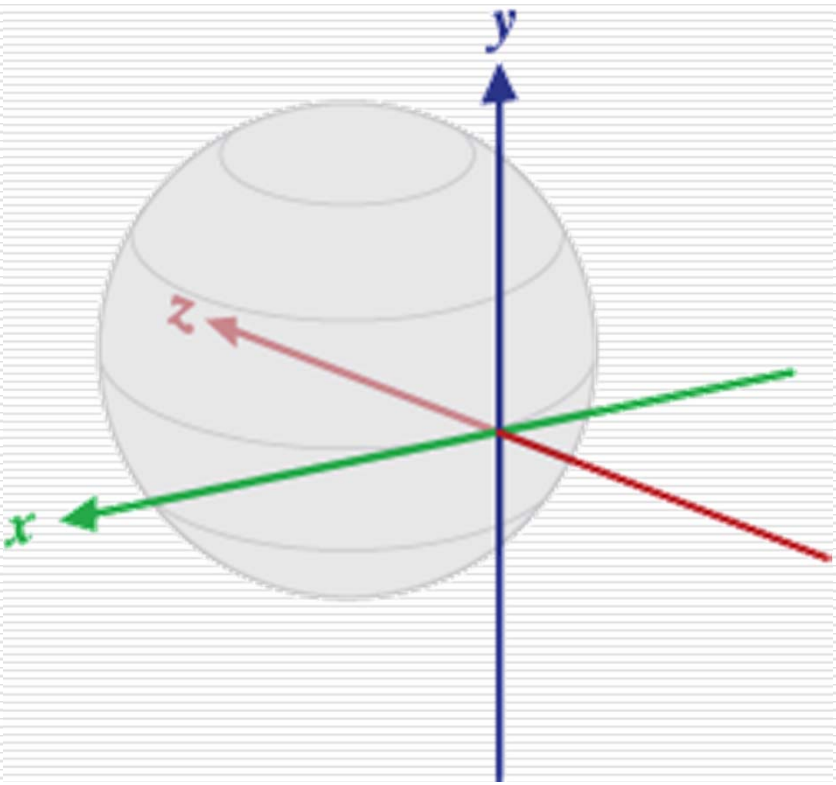
即Y轴与Z轴的向量积Y•Z，方位是与地球球面相切并且指向地理的西方。

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。

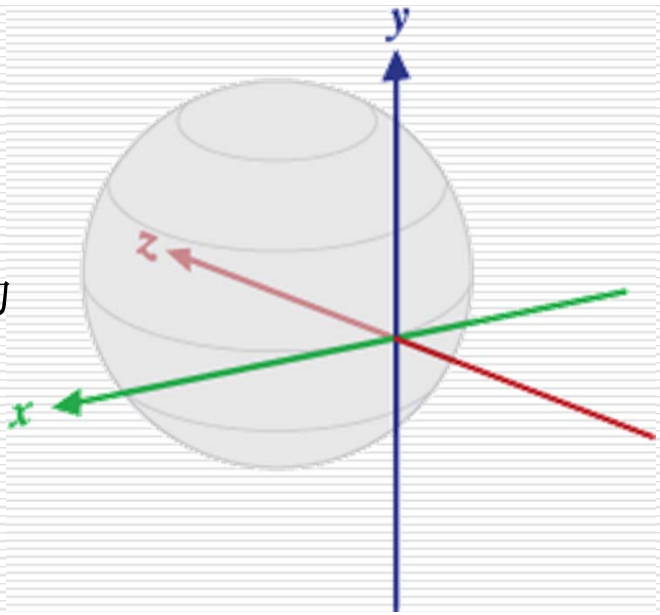
Z轴

为设备所在位置指向地心的方向，垂直于地面。



旋转坐标系

方位传感器与旋转坐标系



当方向传感器感应到方位变化时会返回一个包含变化结果数值的数组，即values[]，数组的长度为3，它们分别代表：

- values[0]——方位角，即手机绕Z轴所旋转的角度。
- values[1]——倾斜角，专指绕X轴所旋转的角度。
- values[2]——翻滚角，专指绕Y轴所旋转的角度。

以上所指明的角度都是逆时针方向的。

获取设备上传感器种类

为了获取当前手机上已连接的传感器清单，需要借助于SensorManager的getSensorList()方法，首先需要获取一个SensorManager类的实例，方法如下：

```
private SensorManager mSensorManager;
mSensorManager = (SensorManager)getSystemService(SENSOR_SERVICE);
```

在获取了当前系统的SensorManager类的对象后，就可以通过其getSensorList()方法来获取相应的传感器清单了，方法如下：

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

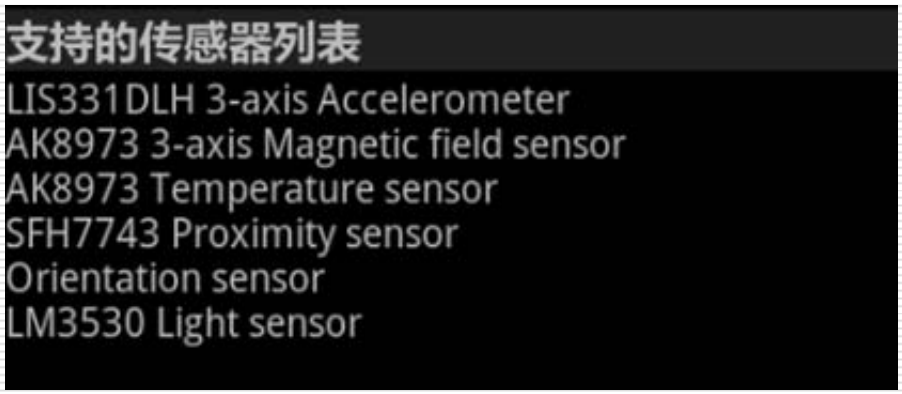
获取了传感器清单后，通过如下代码将每个传感器的名称依次显示到TextView上：

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

获取的传感器列表

SUN YAT-SEN UNIVERSITY

下面是通过前面代码获取到的一个传感器列表：



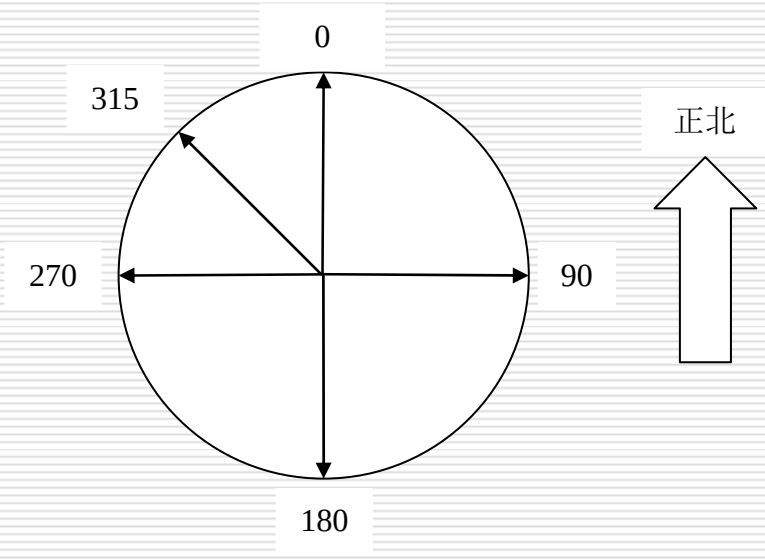
从结果中可以看出，该款真机支持了如下型号的共六种类型的传感器：

- LIS331DLH 3-axis Accelerometer——加速度传感器；
- AK8973 3-axis Magnetic field sensor——磁场传感器；
- AK8973 Temperature sensor——温度传感器；
- SFH7743 Proximity sensor——邻近度传感器；
- Orientation sensor——方位传感器；
- LM3530 Light sensor——亮度传感器。

利用传感器实现指南针功能

功能实现分析

在指南针应用中，需要关注的是手机绕旋转坐标系Z轴所旋转的角度，也就是传感器所传回的values[0]值，该values[0]的值即代表了手机当前已经绕Z轴所旋转的角度，这个角度以正北方向为基准，其返回的值如下图。假定图中右方箭头所指方向为正北，左方圆形中的箭头所指的是手机（传感器）所朝的方向，数值则是传感器返回的values[0]值。



利用传感器实现指南针功能

- 为Activity实现SensorEventListener接口，在类中实现如下两个方法：

```
public void onAccuracyChanged(Sensor sensor, int accuracy) {}

public void onSensorChanged(SensorEvent event) {}
```

- 为当前的Activity注册需要使用的传感器，通过如下的方式获取到系统默认的方位传感器的实例：

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

mOrientation = mSensorManager.getDefaultSensor(Sensor.TYPE_ORIENTATION);
```

利用传感器实现指南针功能

- 完成传感器事件监听器和传感器的注册工作，只有注册了之后，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：

```
protected void onResume() {
    super.onResume();
    mSensorManager.registerListener(this, mOrientation,
            SensorManager.SENSOR_DELAY_UI);
}
protected void onPause() {
    super.onPause();
    mSensorManager.unregisterListener(this);
}
```

利用传感器实现指南针功能

指南针使用ImageView来实现，而指南针旋转则使用了RotateAnimation类，该类专用于定义旋转图像操作，它的一个构造方法如下：

RotateAnimation(float fromDegrees, **float** toDegrees, **int** pivotXType, **float** pivotXValue, **int** pivotYType, **float** pivotYValue**)**

构造方法包含了六个参数，它们的含义分别如下：

- fromDegrees：即该段旋转动画的起始度数；
- toDegrees：旋转的终点度数；
- pivotXType：这个参数用于指定其后的pivotXValue的类型，即说明按何种规则来解析pivotXValue数值，目前包括三种类型即Animation.ABSOLUTE（绝对数值，即pivotXValue为坐标值）、Animation.RELATIVE_TO_SELF（相对于自身的位置，如本示例中的ImageView，当pivotXValue为0.5时表示旋转的轴心X坐标在图形的X边的中点）、Animation.RELATIVE_TO_PARENT（相对于父视图的位置）；
- pivotXValue：即动画旋转的轴心的值，对它的解析依赖于前面的pivotXType指定的类型；
- pivotYType：类似于pivotXType，只是这个参数代表的为Y轴；
- pivotYValue：类似于pivotXValue，只是这个参数代表的为Y轴。

利用传感器实现指南针功能

构造本例中的RotateAnimation对象：

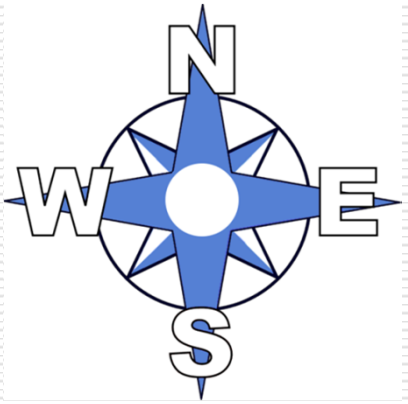
```
1. RotateAnimation ra;
2. ra = new RotateAnimation(currentDegree, targetDegree,
3. Animation.RELATIVE_TO_SELF, 0.5f,
4. Animation.RELATIVE_TO_SELF, 0.5f);
5. ra.setDuration(200);          //在200毫秒之内完成旋转动作
```

如上段代码，第02~04行定义了该旋转动画的属性，即以执行该动画的对象的正中心为旋转轴进行旋转，第05行则设定了完成整个旋转动作这个过程的时间。

有了这个ra对象，ImageView对象通过方法：

```
ImageView.startAnimation(ra);
```

利用传感器实现指南针功能



- 需要找一张图片用于指定各个方向，为了美观和便于使用，示例使用了一张矩形图片，如右图，关于自身中心显得对称，并且该图片中指示北极的箭头是朝正上方的，即可以设定原始图片的旋转度数为0，如果原始图片的北极不是指向正上方也可以使用，但是会为之后的编码引入额外的工作量。准备好了图片资源，将其放入到工程的drawable目录下，并在Activity的onCreate()方法中绑定。

- compass = (ImageView)findViewById(R.id.compass);

利用传感器实现指南针功能

有了compass这个ImageView，就只需要在onSensorChanged()方法中通过传感器传回的数值来定义出需要执行的RotateAnimation ra，并执行compass.startAnimation(ra)就可以实现在传感器每次传回数值时对指南针进行转动，从而实现了指南针的功能。onSensorChanged()的完整代码如下：

```
public void onSensorChanged(SensorEvent event) {
   switch(event.sensor.getType()){
   case Sensor.TYPE_ORIENTATION: {
      //处理传感器传回的数值并反映到图像的旋转上，
      //需要注意的是由于指南针图像的旋转是与手机（传感器）相反的，
      //因此需要旋转的角度为负的角度（-event.values[0]）
      float targetDegree = -event.values[0];
      rotateCompass(currentDegree, targetDegree);
      currentDegree = targetDegree;
      break;
   }
   default:
      break;
   }
}
//…后页续
```

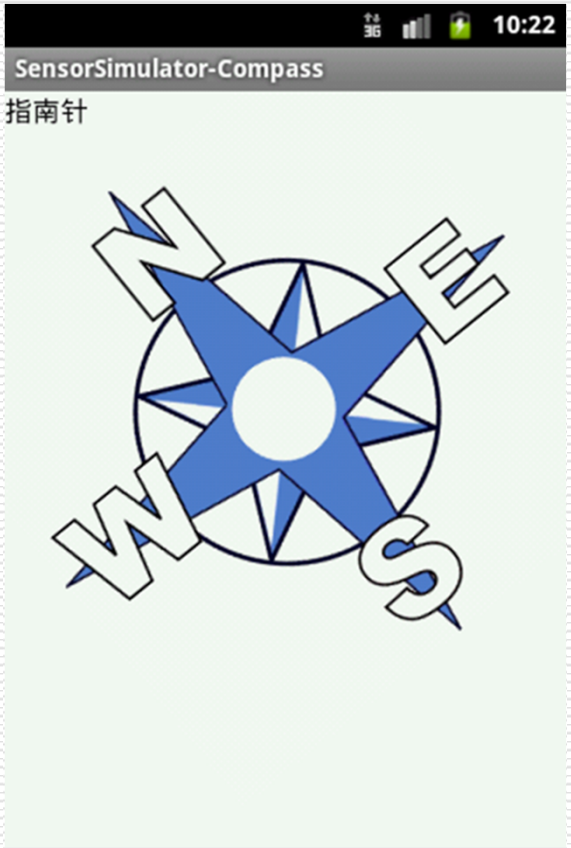
利用传感器实现指南针功能

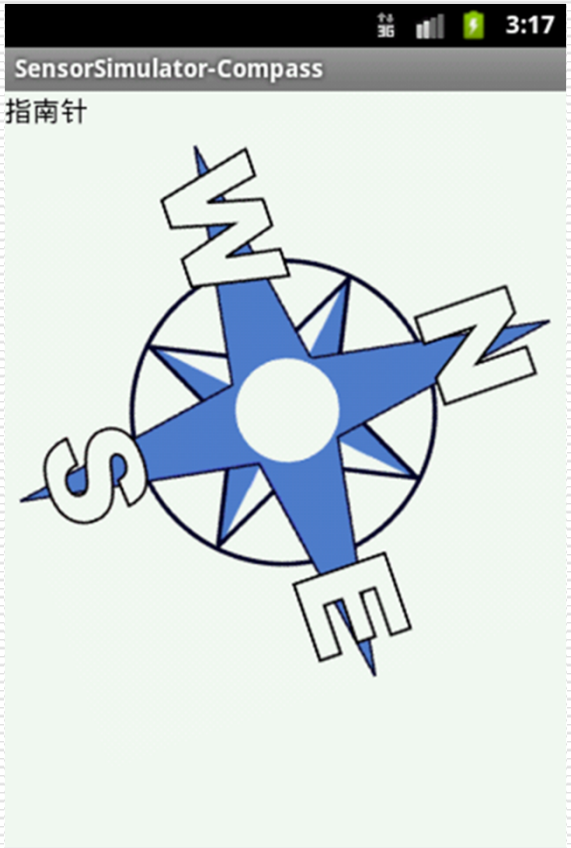
有了compass这个ImageView，就只需要在onSensorChanged()方法中通过传感器传回的数值来定义出需要执行的RotateAnimation ra，并执行compass.startAnimation(ra)就可以实现在传感器每次传回数值时对指南针进行转动，从而实现了指南针的功能。

onSensorChanged()的完整代码如下：

```
//接上页
/**
 * 以指南针图像中心为轴旋转，从起始度数currentDegree旋转至targetDegree
 */
private void rotateCompass(float currentDegree, float
targetDegree){
    RotateAnimation ra;
    ra = new RotateAnimation(currentDegree, targetDegree,
    Animation.RELATIVE_TO_SELF, 0.5f,
    Animation.RELATIVE_TO_SELF, 0.5f);
    ra.setDuration(200);              //在200毫秒之内完成旋转动作
    compass.startAnimation(ra);   //开始旋转图像
}
```

指南针实现效果


10:22
SensorSimulator-Compass
指南针
N
E
W
S


3:17
SensorSimulator-Compass
指南针
W
N
S
E

在模拟器上开发传感器应用

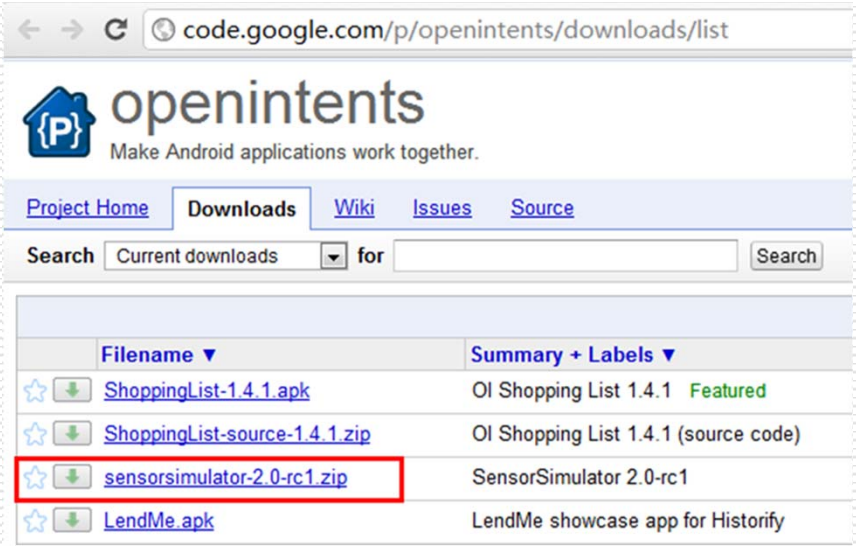
SUN YAT-SEN UNIVERSITY

SensorSimulator

SensorSimulator能够仅仅通过鼠标和键盘就能够实时的仿真出各种传感器的数据，在最新的SensorSimulator版本中甚至还支持了仿真电池电量状态、仿真GPS位置的功能，还能够“录制”真机的传感器在一段时间内的变化情况，以便于为开发者分析和测试提供材料。

OpenIntents项目的下载地址在

http://code.google.com/p/openintents/downloads/list

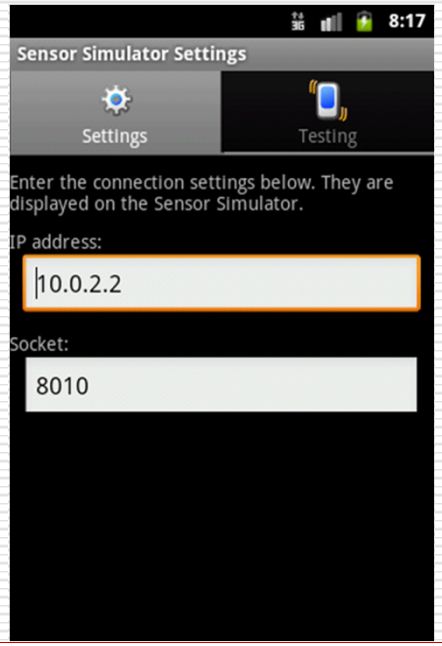


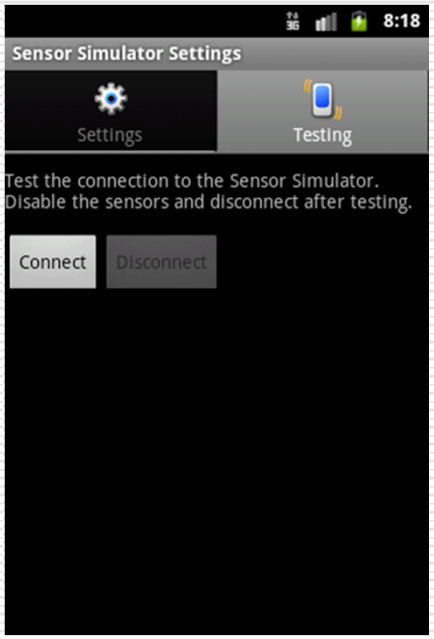
在模拟器上开发传感器应用

- 在模拟器上安装和设置SensorSimulatorSettings
- 通过在命令提示符下键入如下命令来完成安装：

```
adb install SensorSimulatorSettings-2.0-rc1.apk
```

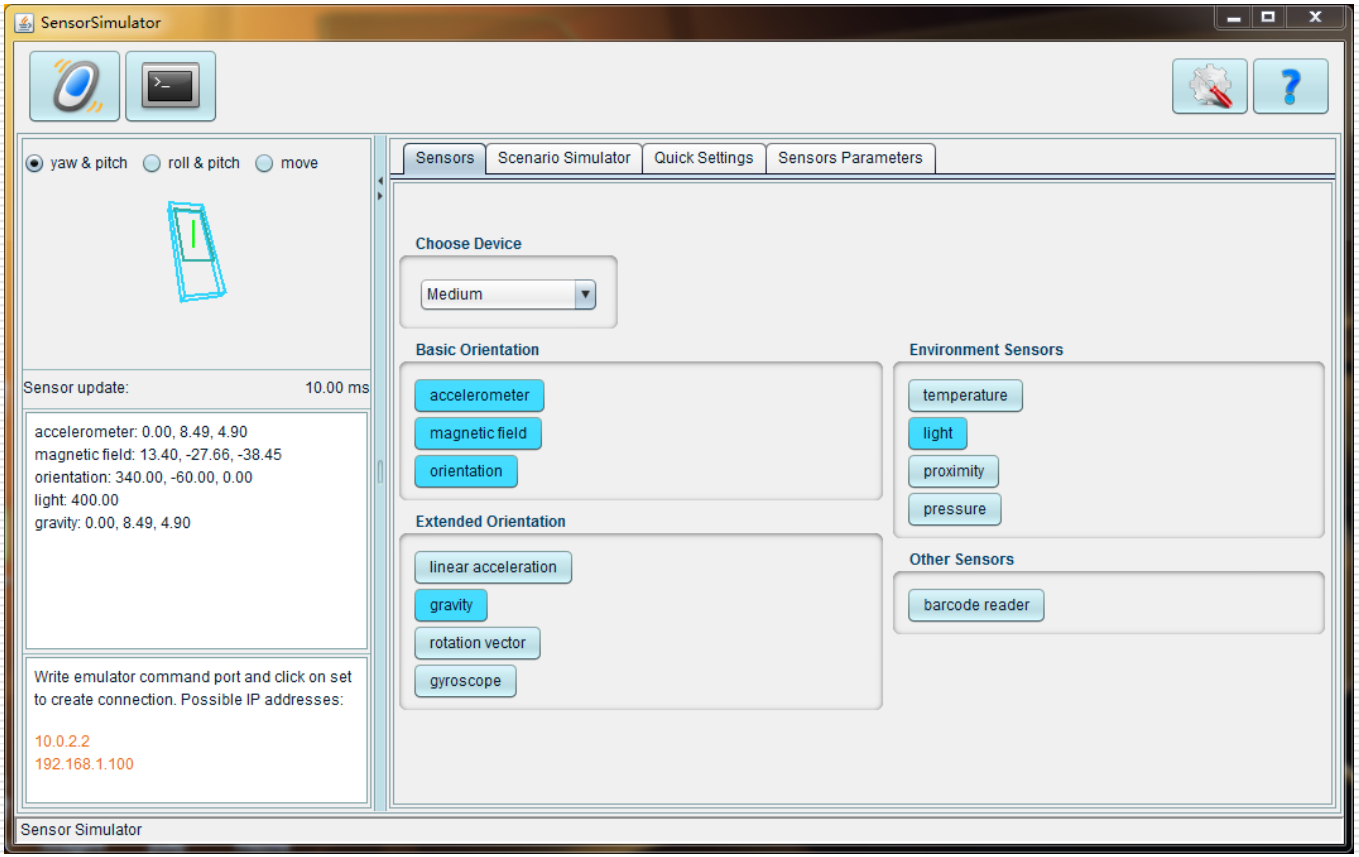
- SensorSimulatorSettings界面





在模拟器上开发传感器应用

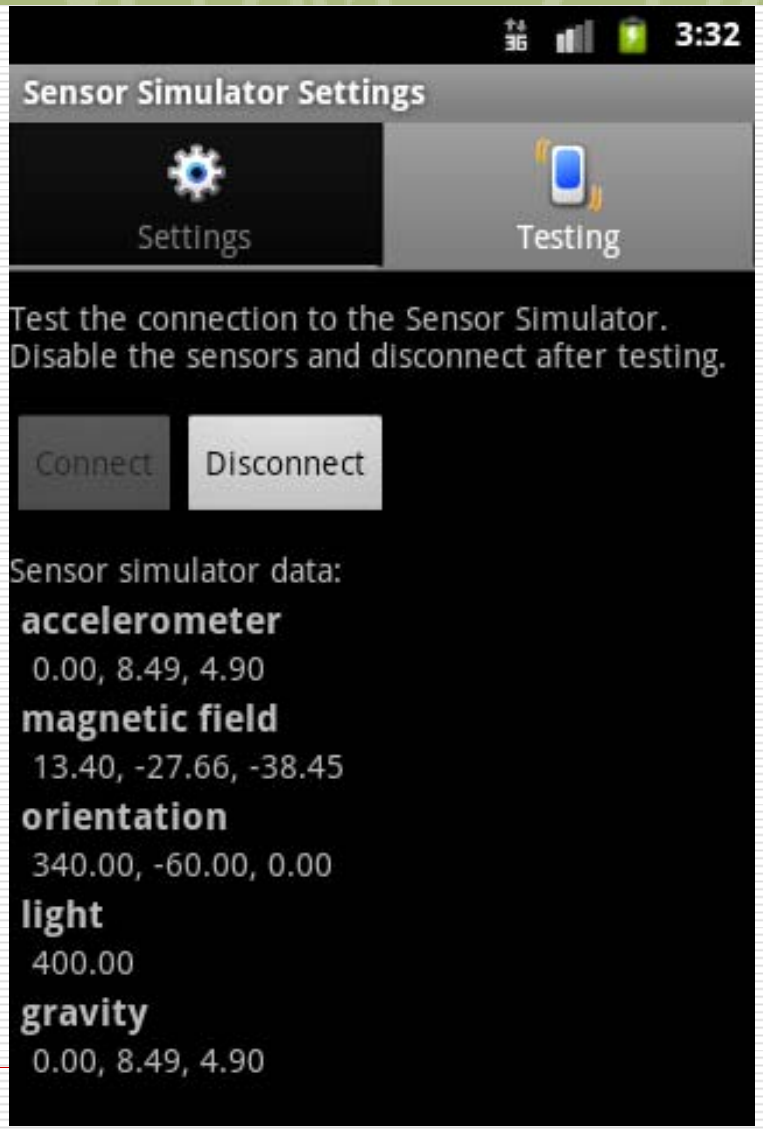
SensorSimulator PC端应用程序



在模拟器上开发传感器应用

SensorSimulator

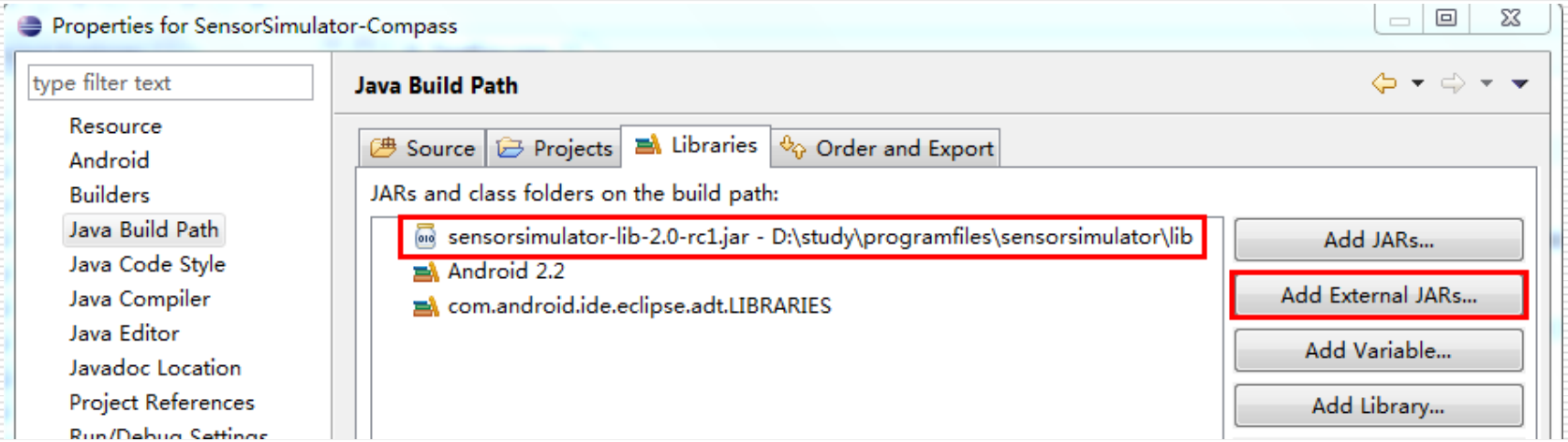
连接模拟器成功



在模拟器上开发传感器应用

修改代码使指南针工作在模拟器上

1. 加载感应器模拟库



2. 引用感应器模拟类库

在模拟器上开发传感器应用

修改代码使指南针工作在模拟器上

（2）引用感应器模拟类库

```
import org.openintents.sensorsimulator.hardware.Sensor;
import org.openintents.sensorsimulator.hardware.SensorEvent;
import org.openintents.sensorsimulator.hardware.SensorEventListener;
import org.openintents.sensorsimulator.hardware.SensorManagerSimulator;
```

需要注意的是，其中第01~03行所导入的类与原本的：

```
import android.hardware.Sensor;
import android.hardware.SensorEvent;
import android.hardware.SensorEventListener;
```

这三条是相冲突的，也正因为如此，模拟器才能够接收到相应的数据，而新导入的SensorManagerSimulator包与原有的SensorManager却是可以并存的，也可以说SensorManager包仍然是新版本项目所需要的包，这是因为它还提供了可供使用的一些具名常量。因此，在导入了四个新的类之后，删除与之冲突的三个类的导入就可以了。

在模拟器上开发传感器应用

修改代码使指南针工作在模拟器上

（3）修改AndroidManifest.xml

由于手机需要通过网络连接到SensorSimulator，因此需要在AndroidManifest.xml加入对网络的使用权限：

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

在模拟器上开发传感器应用

修改代码使指南针工作在模拟器上

（4）修改SensorManager相关代码

另外还需要替换的是获取SensorManager实例的代码，因为SensorManager是用于管理传感器的，而原有的获取其实例的方法显然是不能够是用于仿真出来的传感器的，因此，替换：

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

为如下代码：

```
mSensorManager = SensorManagerSimulator.getSystemService(this, SENSOR_SERVICE);
```

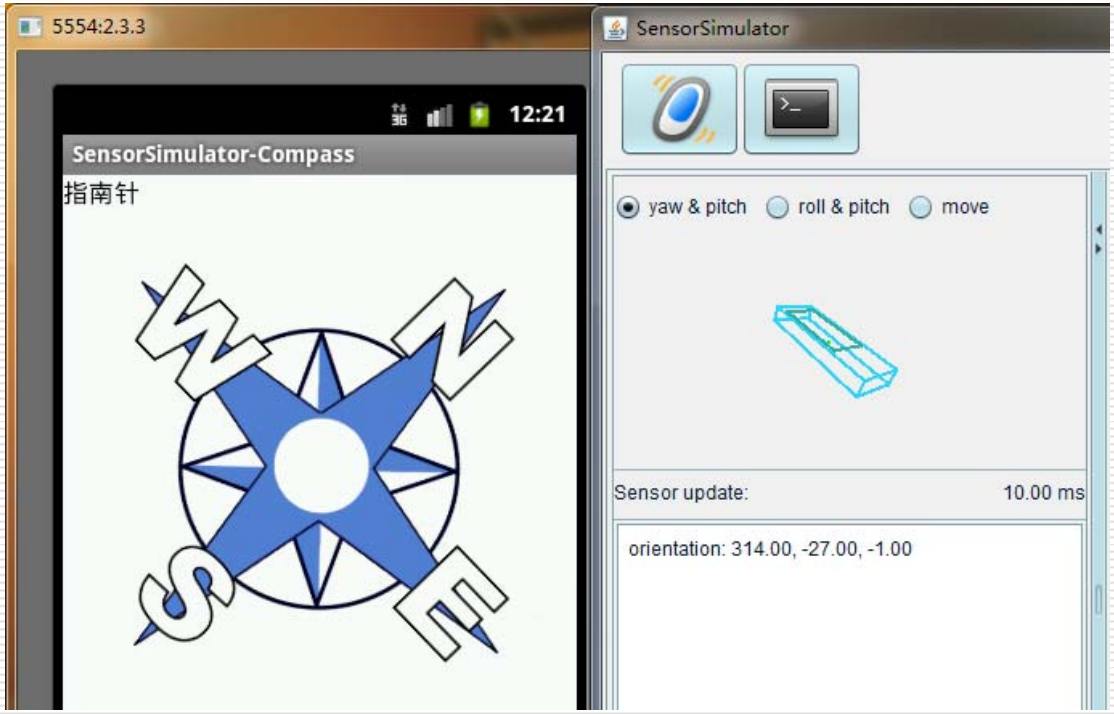
从而获取到用于管理仿真出来的传感器的SensorManager，之后，还需要通过如下方法使得该应用程序连接到SensorSimulator：

```
mSensorManager.connectSimulator();//连接到仿真器
```

在模拟器上开发传感器应用

修改代码使指南针工作在模拟器上

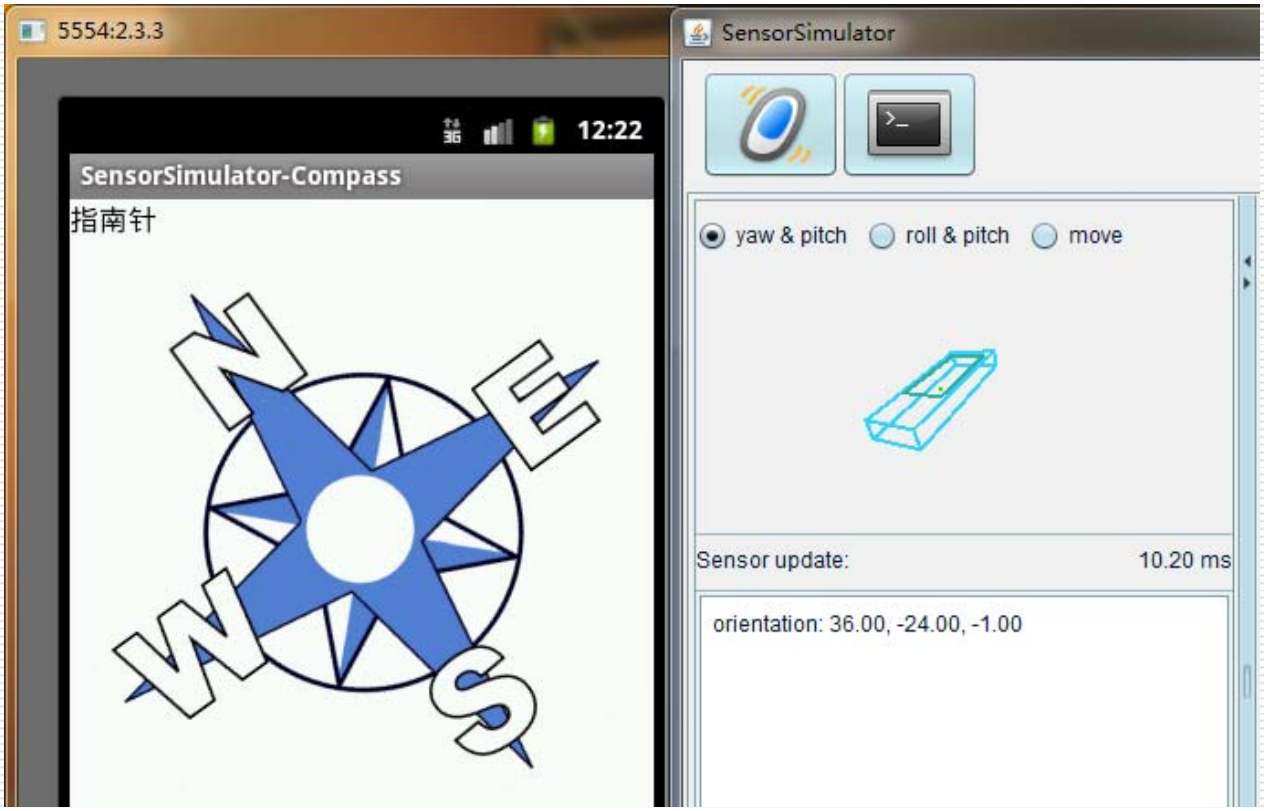
（5）运行测试，状态1



在模拟器上开发传感器应用

修改代码使指南针工作在模拟器上

（5）运行测试，状态2



计步器介绍

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，市面上销售的一些计步器往往还带有其他一些非常丰富的功能，如估算一个人所消耗的能量、估算所走过的距离等等。但这些功能都是建立在准确的测定了人所走的步数之上的，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。下面就在Android平台上来实现一个简易的计步器应用。

计步器所需传感器分析

(1) 加速度传感器——Accelerometer

加速度传感器所测量的是所有施加在设备上的力所产生的加速度（包括了重力加速度）的负值（这个负值是参照世界坐标系而言的，因为默认手机的朝向是向上，而重力加速度则朝下，这里所取为负值可以与大部分人的认知观念相符——即手机朝上时，传感器的数值为正）。加速度所使用的单位是m/s^2，其更新时所返回的SensorEvent.values[]数组的各值含义分别为：

- SensorEvent.values[0]：加速度在X轴的负值；
- SensorEvent.values[1]：加速度在Y轴的负值；
- SensorEvent.values[2]：加速度在Z轴的负值。

(2) 重力加速度传感器——Gravity

重力加速度传感器，其单位也是m/s²，其坐标系与加速度传感器一致。当手机静止时，重力加速度传感器的值和加速度传感器的值是一致的，从SensorSimulator上很容易观察到这一点。

计步器所需传感器分析

SUN YAT-SEN UNIVERSITY

(3) 线性加速度传感器——Linear-Acceleration

这个传感器所传回的数值可以通过如下一个公式清楚的了解：

```
accelerometer = gravity + linear-acceleration
```

如上所述，可知Accelerometer和Linear-Accerleration这两类传感器在本例中几乎可以发挥相同的作用，本例中决定使用Accerlerometer传感器来实现计步器的功能，其实，如果某一款手机不支持Accerlerometer而是支持Linear-Accerleration传感器，只需要通过少量的修改也可使计步器程序变为使用Linear-Accerleration的版本。

计步器功能实现

（1）实现判断走一步的逻辑

```
private static final float GRAVITY = 9.80665f;
private static final float GRAVITY_RANGE = 0.001f;
//存储一步的过程中传感器传回值的数组便于分析
private ArrayList<Float> dataOfOneStep = new ArrayList<Float>();
```

第01、02行代码定义了两个常量，GRAVITY代表了标准的重力加速度值，而GRAVITY_RANGE是一个用于忽略极小的加速度变化的常量，即只要与GRAVITY值相差在该值的范围内时，就认为还是处于标准的重力加速度状态下，可以认为是一种“防抖动”措施；第04行定义了一个ArrayList类型的对象dataOfOneStep用于存储一段连续的传感器数值以供分析使用

计步器功能实现

1. 实现判断走一步的逻辑

   对于走路情景进行简化，走路过程中手机保持在标准姿态，将手机运动简化为在竖直方向来回运动，则Value[2]将随着每一步发生周期性的变化。

2. 注册和使用加速度传感器

3. 将计步结果显示到用户界面

位置服务

SUN YAT-SEN UNIVERSITY

- 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

位置服务

SUN YAT-SEN UNIVERSITY

- 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

位置服务

- 提供位置服务，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
- android.app.Activity.getSystemService()函数代码如下

```
1. String serviceString = Context.LOCATION_SERVICE;
2. LocationManager LocationManager = (LocationManager)getSystemService(serviceString);
```

  - 代码第1行的Context.LOCATION_SERVICE指明获取的服务是位置服务
  - 代码第2行的getSystemService()函数，可以根据服务名称获取Android提供的系统级服务

- 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

位置服务

- 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
- getLastKnowLocation()方法获取当前位置
- LocationManager支持的定位方法有两种
  1. **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
  2. **网络定位：**提供的位置信息精度差，但速度较GPS定位快

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.

- 在指定LocationManager的定位方法后，则可以调用getLastKnowLocation()方法获取当前的位置信息

位置服务

- 以使用GPS定位为例，获取位置信息的代码如下

```
1. String provider = LocationManager.GPS_PROVIDER;
2. Location location = locationManager.getLastKnownLocation(provider);
```

  - 代码第2行返回的Location对象中，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

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

位置服务

- 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

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

  - 第1个参数是定位的方法，GPS定位或网络定位
  - 第2个参数是产生位置改变事件的时间间隔，单位为微秒
  - 第3个参数是距离条件，单位是米
  - 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

位置服务

- 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

实现locationListener的代码如下

```
1.  LocationListener locationListener = new LocationListener(){
2.          public void onLocationChanged(Location location) {
3.          }
4.          public void onProviderDisabled(String provider) {
5.          }
6.          public void onProviderEnabled(String provider) {
7.          }
8.          public void onStatusChanged(String provider, int status, Bundle extras) {
9.          }
10. };
```

位置服务

- 第2行代码onLocationChanged()在设备的位置改变时被调用
- 第4行的onProviderDisabled()在用户禁用具有定位功能的硬件时被调用
- 第6行的onProviderEnabled()在用户启用具有定位功能的硬件时被调用
- 第8行的onStatusChanged()在提供定位功能的硬件的状态改变时被调用，如从不可获取位置信息状态到可以获取位置信息的状态，反之亦然

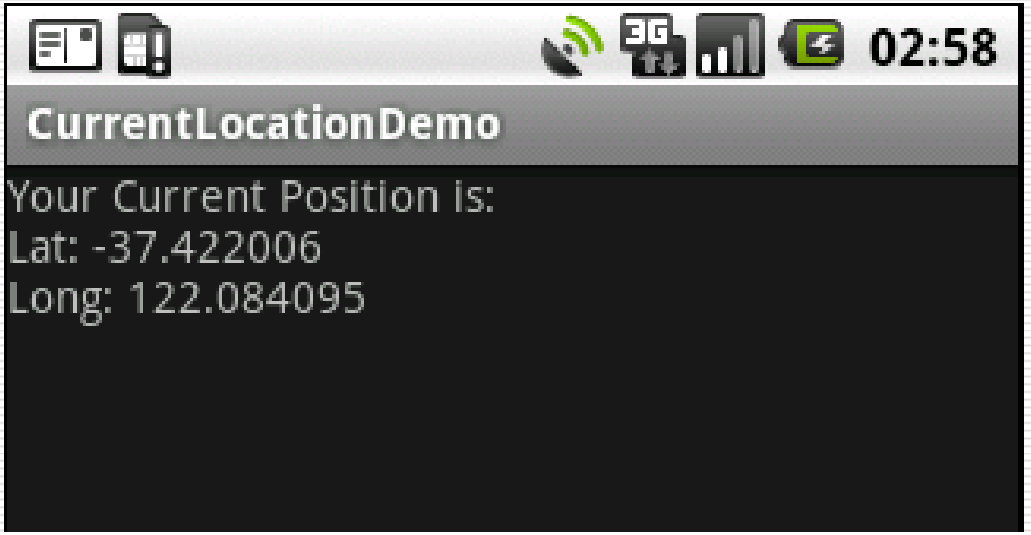
位置服务

- 为了使GPS定位功能生效，还需要在AndroidManifest.xml文件中加入用户许可
- 实现代码如下

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

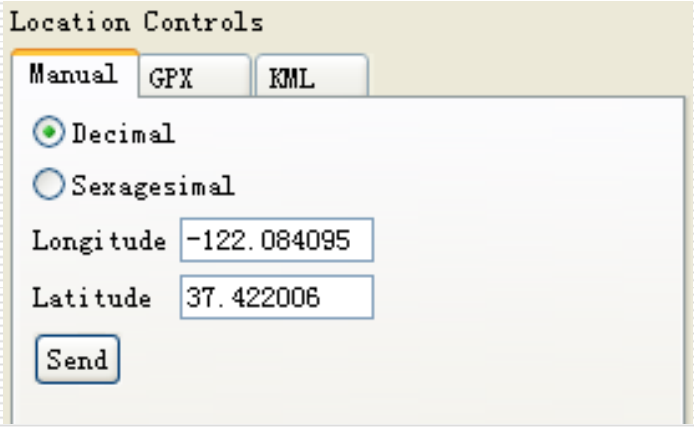
位置服务

- CurrentLocationDemo是一个提供位置服务的基本示例，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



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
  - 打开DDMS中的模拟器控制，在Location Controls中的Longitude和Latitude部分输入设备当前的经度和纬度，然后点击Send按钮，就将虚拟的位置信息发送到Android模拟器中



位置服务

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.

Questions?