#15. 特色开发（NDK,传感器）

学习目标

- Android NDK
- 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 NDK

- Android 应用通常都是使用Java编写，但是有时候开发者需要通过直接在 Android原生交互层（Native Interface）编写代码来管理内存和保证性能， 这时候就不适合使用Java，更适合使用C/C++编写

- Android原生开发工具包（Android provides Native Development Kit ,NDK）是一个工具集，集成了 Android 的交叉编译环境，并提供了一 套比较方便的 Makefile ，可以帮助开发者快速开发 C 或是 C++ 的动态库， 并自动的将 so 和 java 程序打包成 apk ，极大地减轻了开发人员的工作。

- Android NDK不仅允许开发者在Android中使用C/C++代码，而且为开发者提供了平台库以管理原生活动和获取物理设备组件，例如传感器和触控 输入。

Android NDK

- 底层的C/C++代码通过JNI（Java Native Interface）封装成Java接口后才能够在java中调用。

- 只有当应用程序真的是个处理器杀手的时候你才需要使用 NDK，即 NDK适合如下情况：

  1）极其要求设备达到低延迟或者运行计算密集的程序，如游戏或者物理模拟程序。（Android NDK 效率）

  2）重用开发者的C/C++库。（Android NDK 的可移植性）

- 设计的算法需要利用 DalvikVM 中所有的处理器资源，则原生运行较为有利。

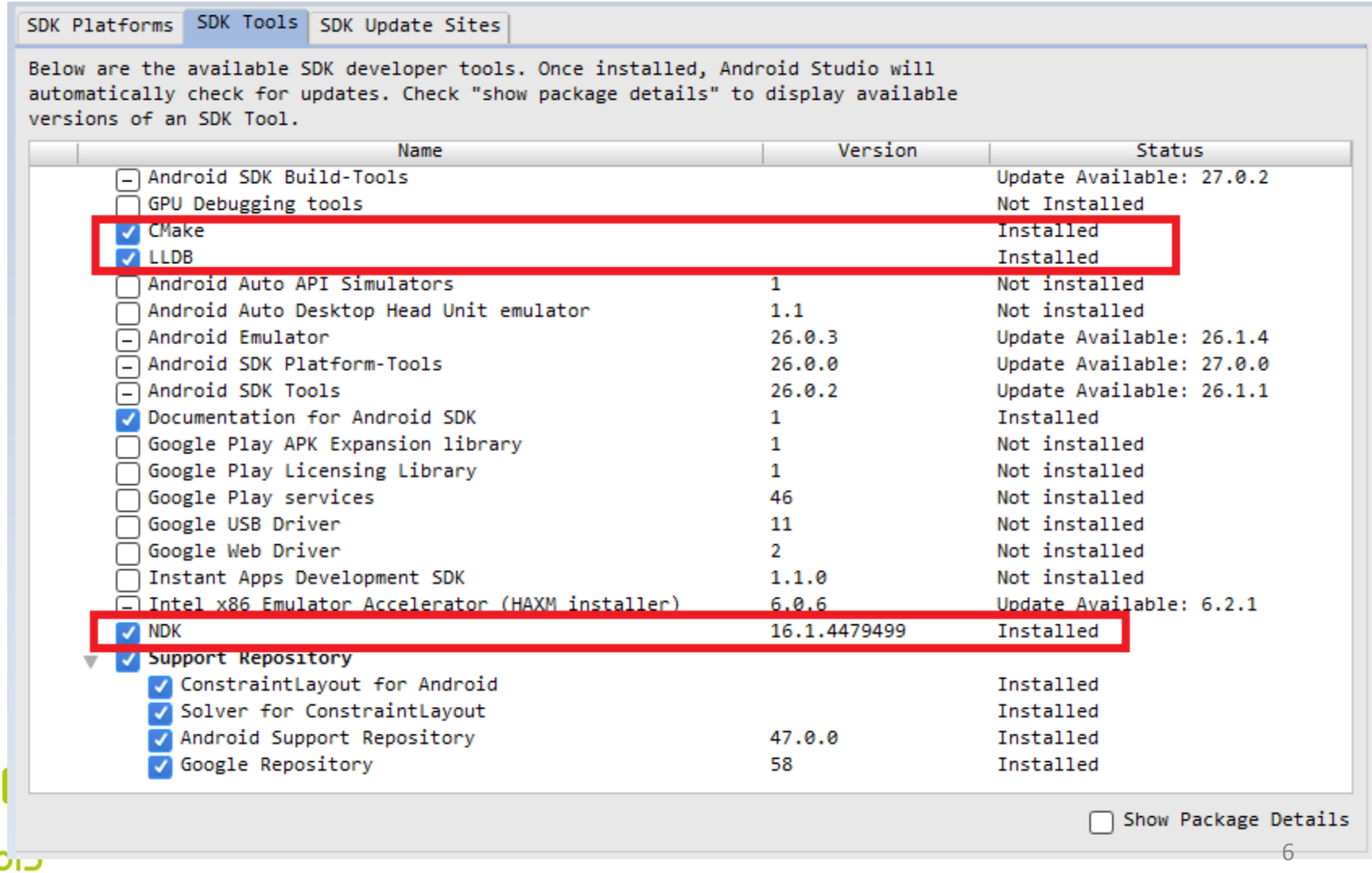
Android NDK

在Android中使用原生库的操作其实很简单，主要步骤就是定义好c/c++的头 文件和源文件，通过ndk编译得到.so动态库，再在java文件中载入.so动态库， 就能够在java文件中调用原生函数了。

下面以一个简单的例子说明如何进行NDK开发：

1. 首先需要配置NDK开发环境。开发NDK需要以下三个组件：Android NDK、 Cmake、LLDB（用于调试原生代码）。这三个组件可以通过SDK Manager 安装。

Android NDK


SDK Platforms
SDK Tools
SDK Update Sites
Below are the available SDK developer tools. Once installed, Android Studio will automatically check for updates. Check "show package details" to display available versions of an SDK Tool.
Name
Version
Status
Android SDK Build-Tools
Update Available: 27.0.2
GPU Debugging tools
Not Installed
CMake
Installed
LLDB
Installed
Android Auto API Simulators
1
Not installed
Android Auto Desktop Head Unit emulator
1.1
Not installed
Android Emulator
26.0.3
Update Available: 26.1.4
Android SDK Platform-Tools
26.0.0
Update Available: 27.0.0
Android SDK Tools
26.0.2
Update Available: 26.1.1
Documentation for Android SDK
1
Installed
Google Play APK Expansion library
1
Not installed
Google Play Licensing Library
1
Not installed
Google Play services
46
Not installed
Google USB Driver
11
Not installed
Google Web Driver
2
Not installed
Instant Apps Development SDK
1.1.0
Not installed
Intel x86 Emulator Accelerator (HAXM installer)
6.0.6
Update Available: 6.2.1
NDK
16.1.4479499
Installed
Support Repository
ConstraintLayout for Android
Installed
Solver for ConstraintLayout
Installed
Android Support Repository
47.0.0
Installed
Google Repository
58
Installed
Show Package Details

Android NDK

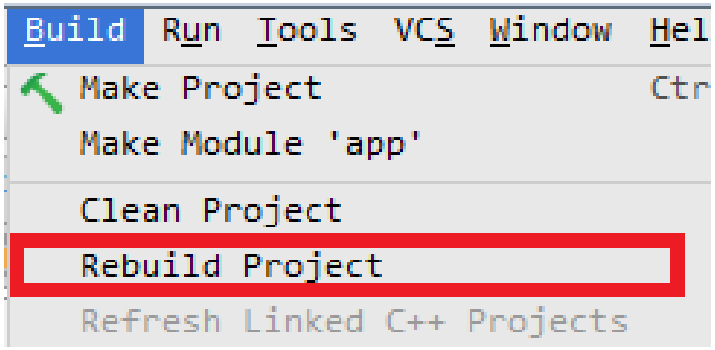
2. 创建一个新项目HelloNDK后，新建一个类JniUtils来实现native方法：

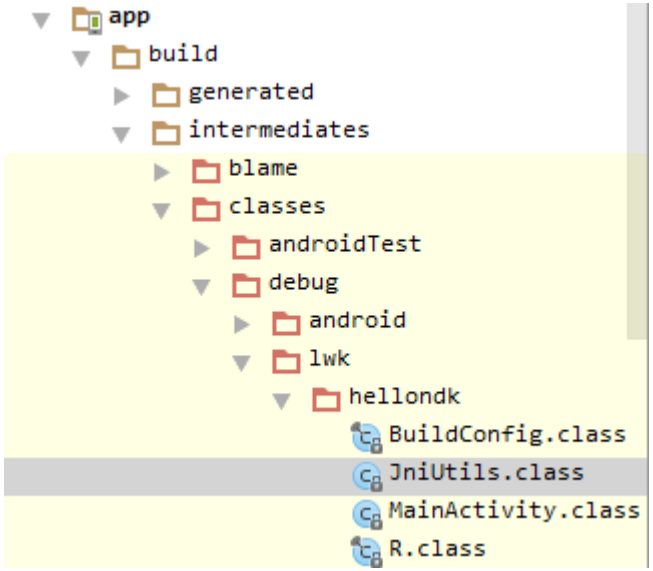
```
public class JniUtils {
    public static native String getStringFromC();
}
```

在JniUtils类中调用了我们在C文件中定义的函数getStringFromC()

3.通过重新编译项目，在 app/build/intermediates /classes/debug/com/.../hellondk下得到一个 JniUtils.class文件。

Android NDK





4. 然后在命令窗口定位到classes\debug目录下，执行命令javah –jni packageNameOfyourproject.hellondk.JniUtils，这会在debug目录下生成一个头文件：packageNameOfyourproject_hellondk_JniUtils.h。在这个头文件中，声明了我们在JniUtils中调用的getStringFromC这个函数。

Android NDK

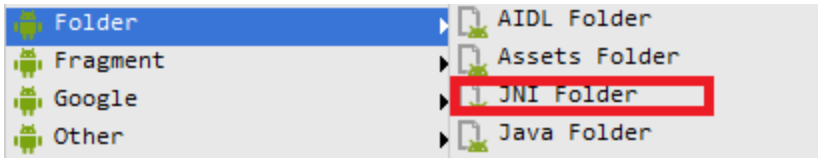
如果出现找不到类文件错误，可以使用-classpath参数将当前路径替换classpath环境变量中指定的路径，具体指令为javah –jni –classpath . packageNameOfyourproject.hellondk.JniUtils

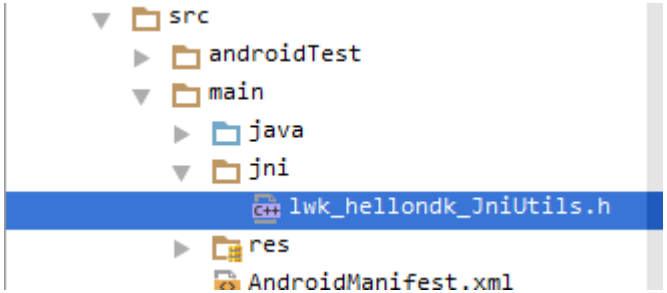
```
C:\Users\Administrator\AndroidStudioProjects\HelloNDK\app\build\intermediates\classes\debug>javah -classpath . -jni lwk.hellondk.JniUtils
```

```
/* DO NOT EDIT THIS FILE - it is machine generated */
#include <jni.h>
/* Header for class lwk_hellondk_JniUtils */

#ifndef _Included_lwk_hellondk_JniUtils
#define _Included_lwk_hellondk_JniUtils
#ifdef __cplusplus
extern "C" {
#endif
/*
 * Class:     lwk_hellondk_JniUtils
 * Method:    getStringFromC
 * Signature: ()Ljava/lang/String;
 */
JNIEXPORT jstring JNICALL Java_lwk_hellondk_JniUtils_getStringFromC
  (JNIEnv *, jclass);

#ifdef __cplusplus
}
```

Android NDK

5.在 src/main下新建文件夹jni,把生成的.h文件移动到该目录





6. 配置gradle文件 1) 修改build.gradle配置，在defaultConfig里面配置ndk

```
defaultConfig {
    applicationId "lwk.hellondk"
    minSdkVersion 15
    targetSdkVersion 26
    versionCode 1
    versionName "1.0"
    testInstrumentationRunner "android.support.test.runner.AndroidJUnitRunner"

    ndk {
        moduleName "NdkJniDemo"          //生成的so名字
        abiFilters "armeabi", "armeabi-v7a", "x86" //输出指定三种abi体系结构下的so库，目前可有可无
    }
}
```

Android NDK

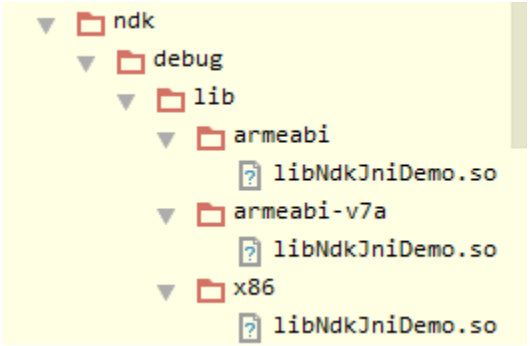
2)在gradle.properties里面配置可以使用被启用的ndk

```
android.useDeprecatedNdk=true
```

7. 在jni目录下新建一个JniUtils.cpp源文件实现getStringFromC函数

```
#include "lwk_hellondk_JniUtils.h"
JNIEXPORT jstring JNICALL Java_lwk_hellondk_JniUtils_getStringFromC
        (JNIEnv *env, jclass obj){
    return env->NewStringUTF("这里是来自c的string");
    }
```

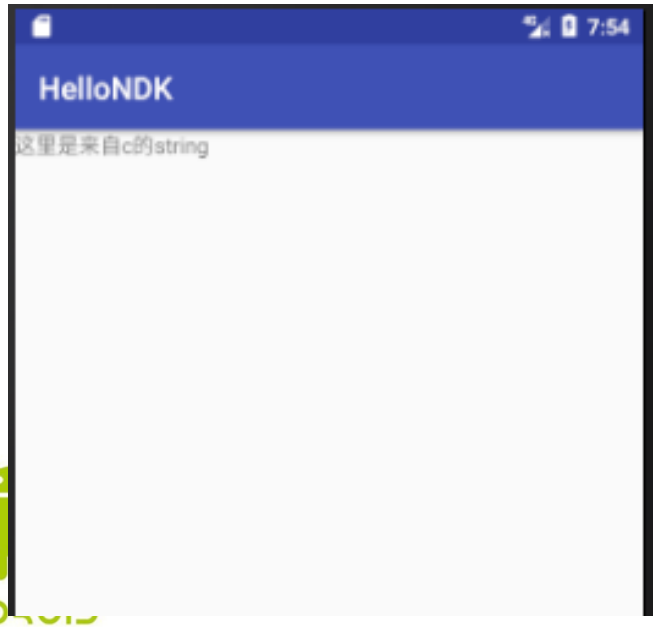
8. 重新编译项目生成不用平台下的 动态库文件xxx.so文件



Android NDK

9. 最后在在JniUtils里面，加载so文件，就可以调用原生函数 getStringFromC ()。

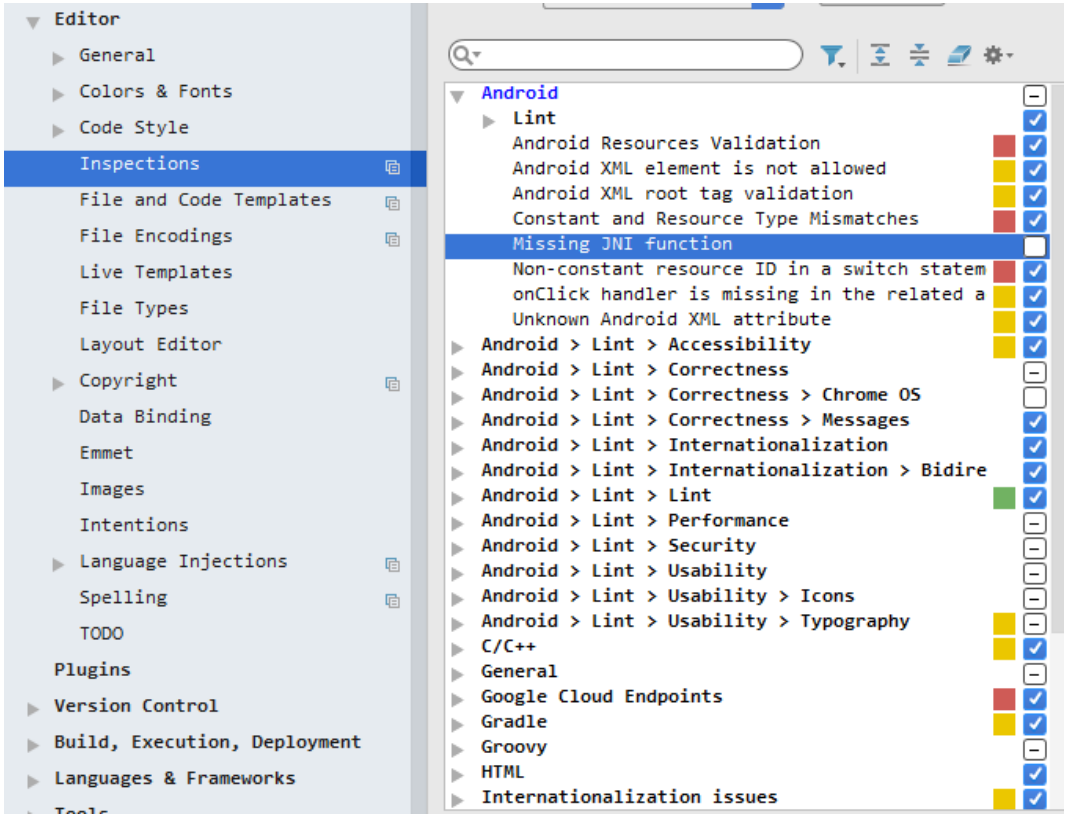
```
public class JniUtils {

    public static native String getStringFromC();
    static{
        System.load("NdkJniDemo");
    }
```




```
package lwk.hellondk;

import ...

public class MainActivity extends AppCompatActivity {
    private TextView textView;
    JniUtils jniUtils = new JniUtils();

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

        textView = (TextView)findViewById(R.id.textview);
        textView.setText(jniUtils.getStringFromC());
    }
}
```

Android NDK

注意：通过File>Setting>Editor>Inspections)来选择要进行更改的配置文件，搜索“JNI”并取消选中“Missing JNI function”来解决JNI函数飘红的问题



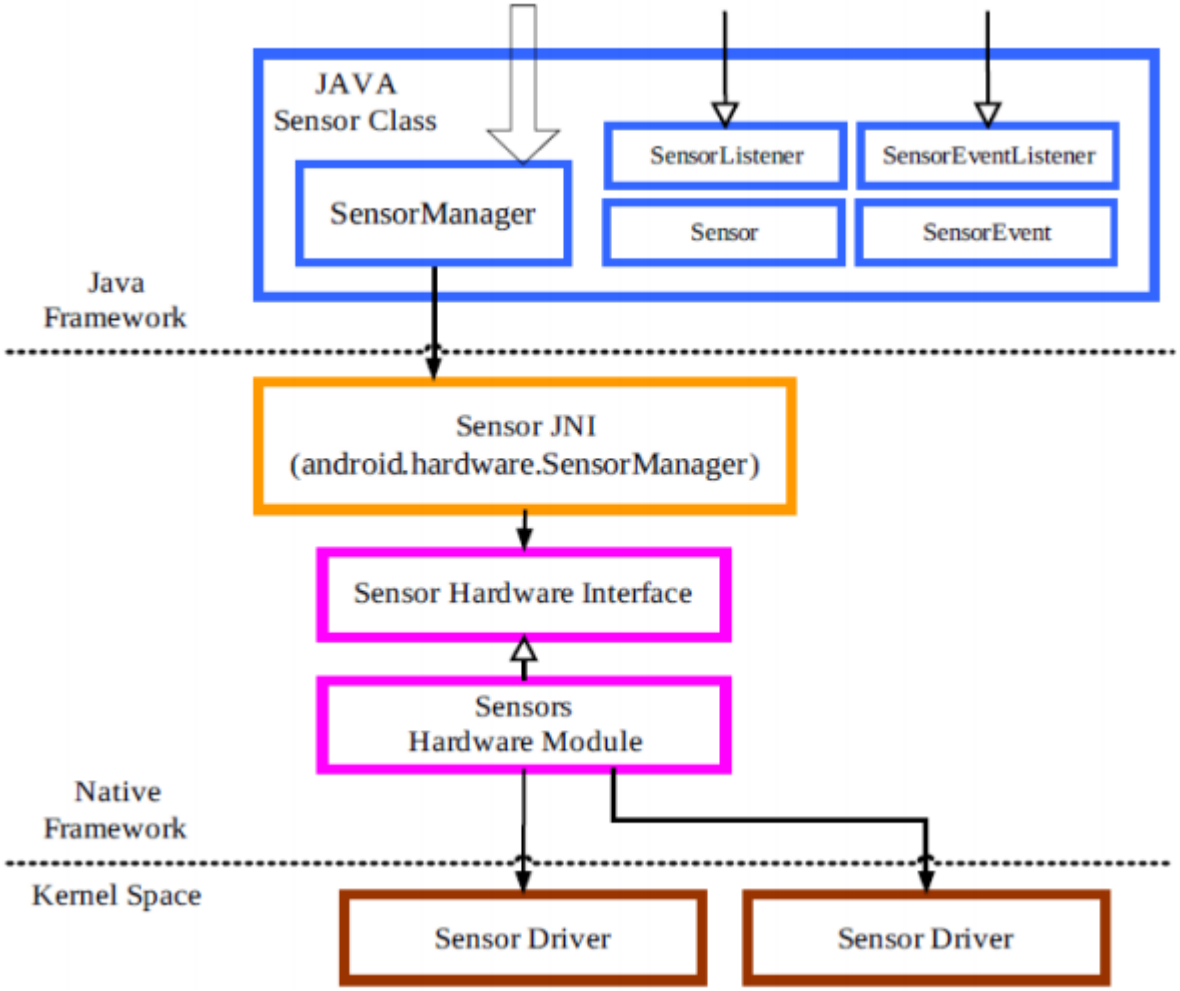
Android 传感器系统概述

- 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**

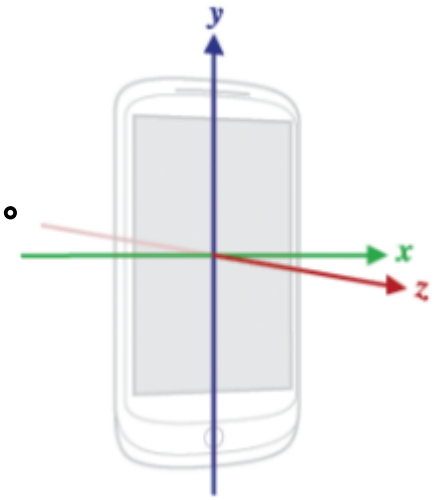
  最快、游戏、普通、用户界面。当应用程序请求特定的采样率时，其实只是对传感 器子系统的一个提示，或者一个建议。不保证特定的采样率可用。

- **准确性**

  高、低、中、不可靠。

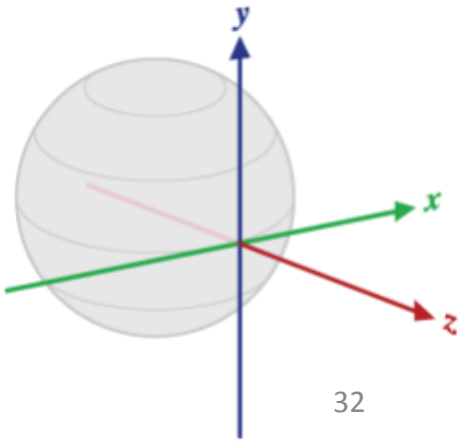
常见的传感器

- 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

https://developer.android.com/guide/topics/sensors/sensors_overview.html

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。

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（Ecompass）。

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。

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，取而代之使用 getRotationMatrix()和getOrientation()方法计算方向数据

```
/**
 * A constant describing an orientation sensor type.
 * <p>See {@link android.hardware.SensorEvent#values SensorEvent.values}
 * for more details.
 *
 * @deprecated use {@link android.hardware.SensorManager#getOrientation
 *             SensorManager.getOrientation()} instead.
 */
@Deprecated
public static final int TYPE_ORIENTATION = 3;
```

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。

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。

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。

其他传感器

- **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4个步骤：

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getSystemServiceu获取传感器句柄
sensorManager = (SensorManager)getSystemService(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

```
//获取默认的加速度传感器
Sensor AccelerometerSensor = sensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
```

传感器系统的使用

3. 实现监听器的 SensorEventListner 接口

```
listener = new SensorEventListener() {
    @Override
    public void onSensorChanged(SensorEvent event) {
        //当传感器数值变化时调用
    }

    @Override
    public void onAccuracyChanged(Sensor sensor, int accuracy) {
        //当传感器精度变化时调用
    }
};
```

4. 注册监听器，监听传感器数据变化

```
//注册监听器
sensorManager.registerListener(listener,AccelerometerSensor,SensorManager.SENSOR_DELAY_UI);
//SENSOR_DELAY_UI:适合普通用户界面UI变化的频率
```

传感器系统的使用

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

```
//获取默认的加速度传感器
Sensor AccelerometerSensor = sensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
```

- 第 2 种：获取某种传感器的列表

```
//获取压力传感器列表
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

对于某一个传感器，它的一些具体信息的获取方法

```
public void onResume(){
    super.onResume();
    //注册监听器
    sensorManager.registerListener(listener,AccelerometerSensor,SensorManager.SENSOR_DELAY_UI);
    //SENSOR_DELAY_UI:适合普通用户界面UI变化的频率
}
public void onPause(){
    super.onPause();
    sensorManager.unregisterListener(listener);
}
```

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

获取设备上传感器

```
public class MainActivity extends AppCompatActivity {

    private TextView textView;
    private SensorManager sensorManager;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        textView = (TextView)findViewById(R.id.textview);

        //通过getSystemServiceu获取传感器句柄
        sensorManager = (SensorManager)getSystemService(SENSOR_SERVICE);

        //获取默认的加速度传感器
        Sensor AccelerometerSensor = sensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
        //获取压力传感器列表
        List<Sensor> pressureSensors = sensorManager.getSensorList(Sensor.TYPE_PRESSURE);
        //获取所有传感器
        List<Sensor> allSensors = sensorManager.getSensorList(Sensor.TYPE_ALL);

        for(Sensor sensor:allSensors){
            textView.append(sensor.getName() + '\n');
        }
    }
}
```

在模拟器 Nexus 5X API 25 上，通过代码获取到一共9种传感器

传感器系统的使用

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（如果不释放感应器会一直工作，即使屏幕关闭系统也不会自动关闭感应器，耗电十分严重）

```
public void onResume(){
    super.onResume();
    //注册监听器
    sensorManager.registerListener(listener,AccelerometerSensor,SensorManager.SENSOR_DELAY_UI);
    //SENSOR_DELAY_UI:适合普通用户界面UI变化的频率
}
public void onPause(){
    super.onPause();
    sensorManager.unregisterListener(listener);
}
```

传感器系统的使用

- 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。

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。

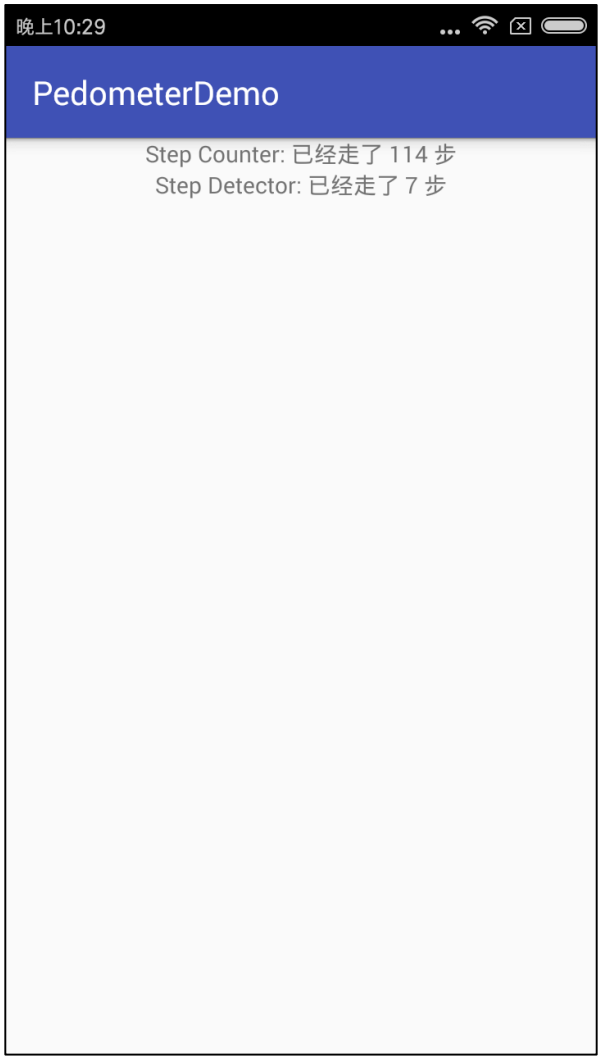
计步器

- 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。

计步器

```
//Step Counter
sensorManager.registerListener(new SensorEventListener(){
    @Override
    public void onSensorChanged(SensorEvent event){
        float steps = event.values[0];
        counter.setText("Step Counter: 已经走了 " + (int)steps + " 步");
    }
    @Override
    public void onAccuracyChanged(Sensor sensor, int accuracy){
    }
}, counterSensor, SensorManager.SENSOR_DELAY_UI);

//Step Detector
sensorManager.registerListener(new SensorEventListener(){
    private int step;
    @Override
    public void onSensorChanged(SensorEvent event){
        if (event.values[0] == 1.0f)
            step++;
        detector.setText("Step Detector: 已经走了 " + (int)step + " 步");
    }
    @Override
    public void onAccuracyChanged(Sensor sensor, int accuracy){
    }
}, detectorSensor, SensorManager.SENSOR_DELAY_UI);
```

计步器


晚上10:29
PedometerDemo
Step Counter: 已经走了 114 步
Step Detector: 已经走了 7 步

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

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

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。 传感器系统的使用位置服务

```
locationManager = (LocationManager)getSystemService(LOCATION_SERVICE); //获取系统服务
```

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

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

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

```
final String provider = LocationManager.GPS_PROVIDER;        //定义定位方法为GPS
```

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

```
//设置时间变化频率
locationManager.requestLocationUpdates(provider,2000,10,locationListener);
```

位置服务

- 4. 定义处理位置变化事件监听类 LocationListener。

```
//定义回调函数
locationListener = new LocationListener() {
    @Override
    public void onLocationChanged(Location location) {
        //在设备的位置改变时调用
        lat = location.getLatitude();
        lon = location.getLongitude();
        latitude.setText("纬度：  " + Double.toString(lat));
        longitude.setText("经度：  " + Double.toString(lon));
    }

    @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调用
    }
```

位置服务

- 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

```
lat = location.getLatitude();
lon = location.getLongitude();
latitude.setText("纬度:  " + Double.toString(lat));
longitude.setText("经度:  " + Double.toString(lon));
```

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

- 注意当使用GPS定位时，最好不要使用getLastKnownLocation方法获得当前位置对象Location，因为该对象可以在onLocationChanged的参数中由系统给予这样就避免了空指针异常。而且更重要的是GPS定位不是一下子就能定位成功的，在90%以上的情况下，getLastKnownLocation返回null

```
//获取缓存位置信息
Location lastKnownLocation = locationManager.getLastKnownLocation(LocationManager.NETWORK_PROVIDER)
//模拟器会返回null
```

- 在获取位置之前，需要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。

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。

位置服务

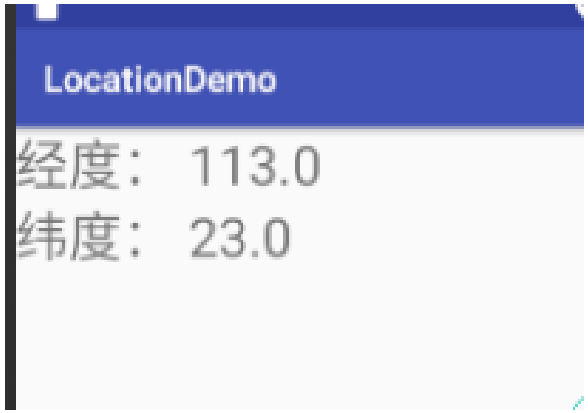
- 如果目标设备是Android5.0 (API 21)及以上，必须声明程序使用android.hardware.location.network or android.hardware.location.gps 硬件 特征。在Android5.0 (API 21)以前，声明ACCESS_COARSE_LOCATION 和 ACCESS_FINE_LOCATION 权限，就包括了使用两种硬件特征。

- 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"/>
<uses-feature android:name="android.hardware.location.network"/>
```

位置服务

- 通过以上步骤，实现了一个LocationDemo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。

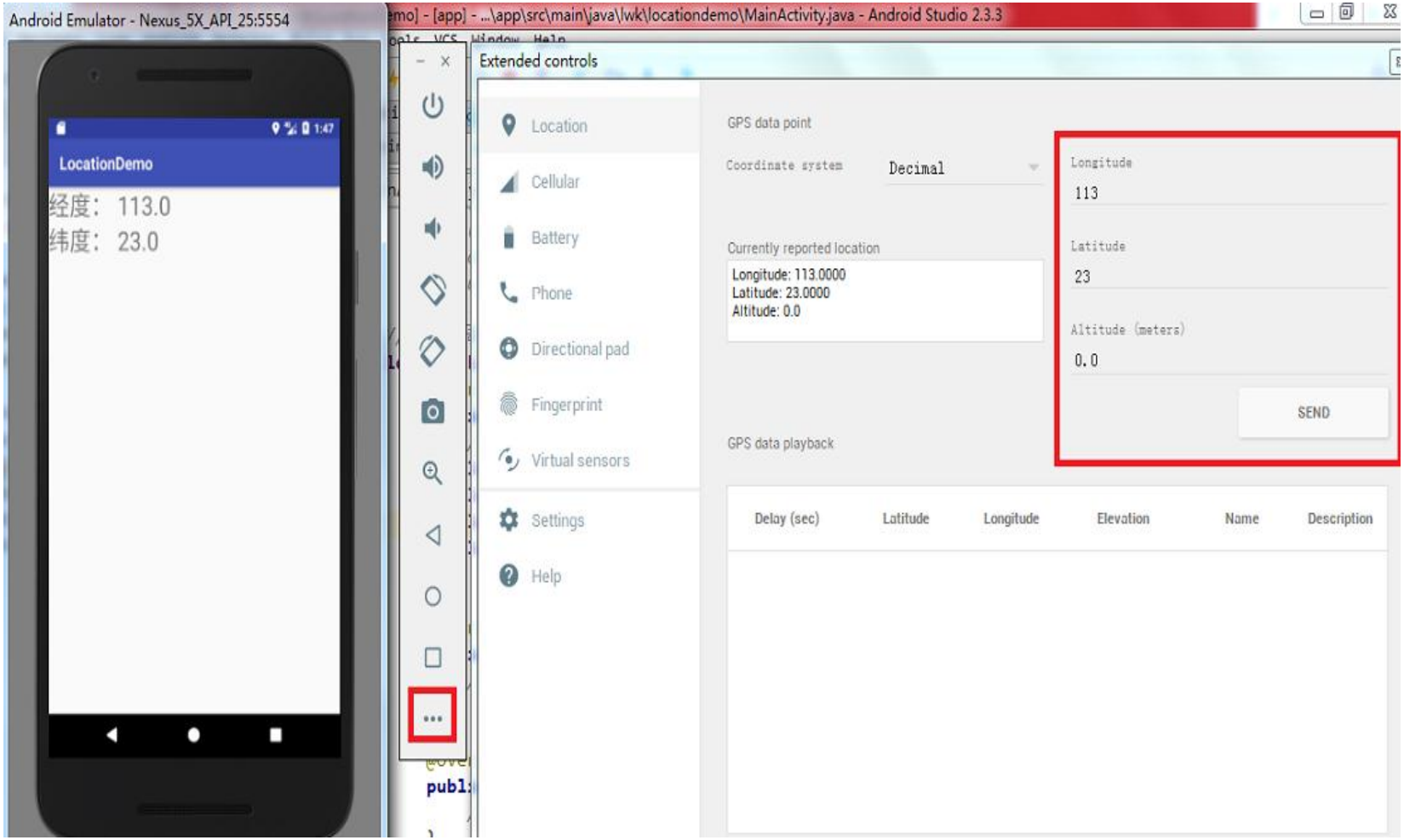


位置服务

- 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。

位置服务


Android Emulator - Nexus_5X_API_25:5554
LocationDemo
经度： 113.0
纬度： 23.0
Extended controls
Location
Cellular
Battery
Phone
Directional pad
Fingerprint
Virtual sensors
Settings
Help
GPS data point
Coordinate system
Decimal
Currently reported location
Longitude: 113.0000
Latitude: 23.0000
Altitude: 0.0
Longitude
113
Latitude
23
Altitude (meters)
0.0
SEND
GPS data playback
Delay (sec)
Latitude
Longitude
Elevation
Name
Description

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。


Questions?