#12 特色应用

学习目标

- Android NDK
- Android传感器系统
- 位置服务
- 地图应用

NDK简介

- 在 Android 上，应用程序的开发，大部分基于 Java 语言来实现。
- 要使用 c 或是 c++ 的程序或库，就需要使用 NDK 来实现。
- NDK 是 Native Development Kit 的简称。它是一个工具集，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，在 Android 上运行。

NDK效率

- 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。
- 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。
- 在 Android 2.2 中，JIT 编译器会提高类似代码的效率。

NDK可移植性

- 若现有的应用程序中有大量的 C 语言代码，使用 NDK 不仅可以加速你的项目的开发进程，也能在Android 和非 Android 项目中保持修改的同步。
- 这对于那些为其他平台而写的 OpenGL ES 应用程序来说尤为如此。

NDK兼容性

- 这是原生代码，在使用时由处理器构架编译。它支持何种处理器构架？
- 目前的 NDK中，只支持 ARMv5TE 和 ARMv7-A 指令集。
- 默认设置下，目标架构被设置为 ARMv5TE，它可以在使用 ARM 芯片的 Android 设备上运行。

NDK使用简介

- Android VM允许java代码通过JNI去调用原生代码（native code）

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.pickertester);
    Log.v("TAG",stringFromJNI());
}

public native String  stringFromJNI();
static {
    System.loadLibrary("hello-jni"); //载入hello-jni库
}
```

NDK使用简介

```
// C代码，将编译成库
#include <string.h>
#include <jni.h>

jstring //返回值
Java_com_example_hellojni_HelloJni_stringFromJNI(
        JNIEnv* env,    //JNI环境
        jobject thiz
)
{
    return (*env)->NewStringUTF(
                        env,
                        "Hello from JNI !");
}
```

Android的传感器系统

- 综　述
- 层次结构
- 硬件抽象层
- 使　用

传感器系统综述

- 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。

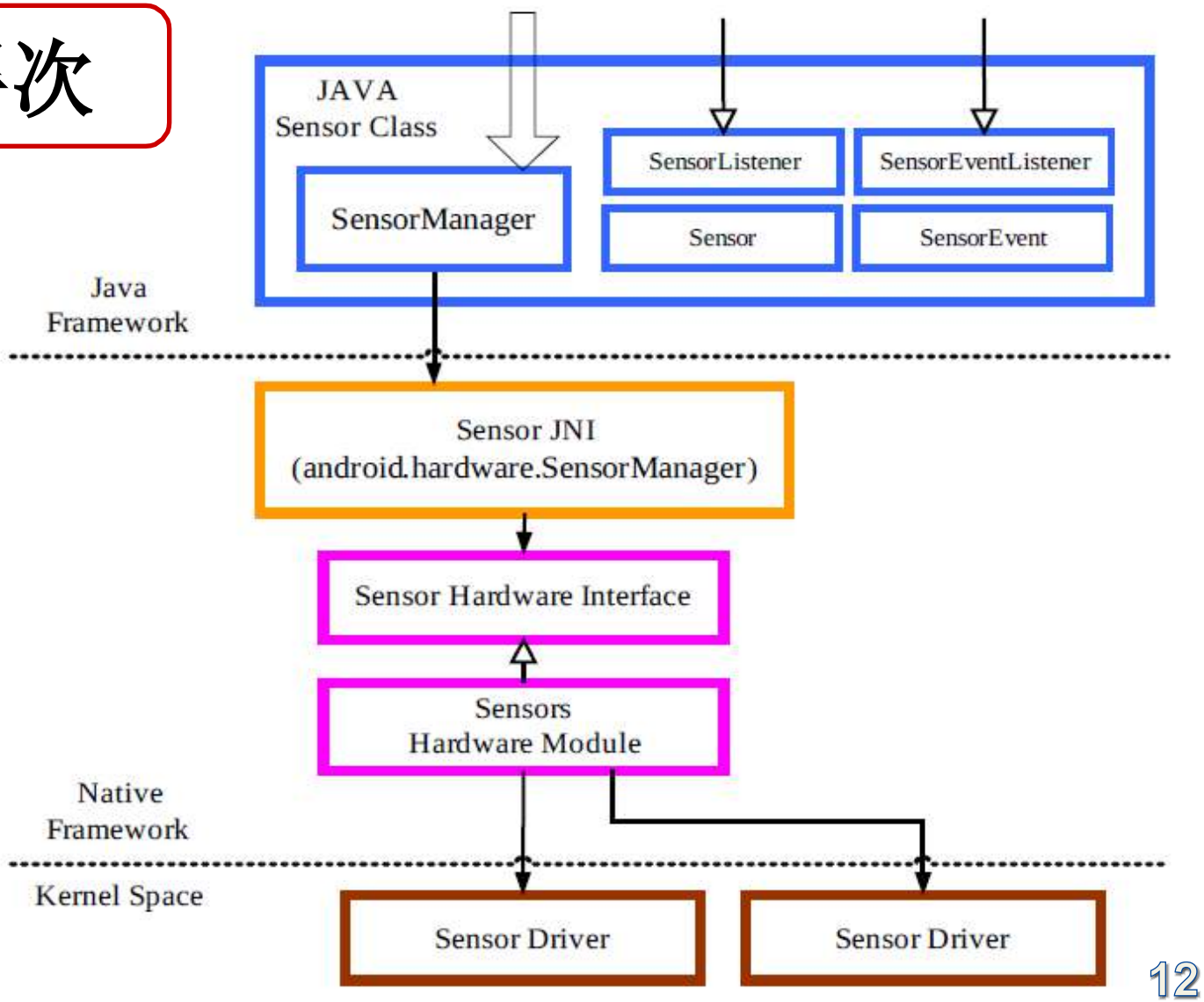
传感器系统综述

传感器种类

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

传感器层次



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

- 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 }
}
```

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.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

- 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。

传感器系统的使用

- Android所有的传感器都归传感器管理器 SensorManager 类管理
- 获取传感器管理器的方法

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

传感器系统的使用

- **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。

传感器系统的使用

- **现阶段Android支持的传感器有8种**

传感器类型常量	内部整数值	中文名称
Sensor.TYPE_ACCELEROMETER	1	加速度传感器
Sensor.TYPE_MAGNETIC_FIELD	2	磁力传感器
Sensor.TYPE_ORIENTATION	3	方向传感器
Sensor.TYPE_GYROSCOPE	4	陀螺仪传感器
Sensor.TYPE_LIGHT	5	环境光照传感器
Sensor.TYPE_PRESSURE	6	压力传感器
Sensor.TYPE_TEMPERATURE	7	温度传感器
Sensor.TYPE_PROXIMITY	8	距离传感器

传感器系统的使用

(1)加速度传感器

- 加速度传感器又叫G-sensor，返回x、y、z三轴的加速度数值。
- 该数值包含地心引力的影响，单位是m/s^2。
- 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。
- 将手机朝下放在桌面上，z轴为-9.81。
- 将手机向左倾斜，x轴为正值。
- 将手机向右倾斜，x轴为负值。
- 将手机向上倾斜，y轴为负值。
- 将手机向下倾斜，y轴为正值。

传感器系统的使用

(2)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）。
- 电子罗盘传感器同时提供下文的方向传感器数据。

传感器系统的使用

(3)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。
- 为了得到精确的角度数据，E-compass需要获取G-sensor的数据，
- 经过计算生产O-sensor数据，否则只能获取水平方向的角度。
- 方向传感器提供三个数据，分别为azimuth、pitch和roll。
- azimuth：方位，返回水平时磁北极和Y轴的夹角，范围为0°至360°。
- 0°=北，90°=东，180°=南，270°=西。
- pitch：x轴和水平面的夹角，范围为-180°至180°。
- 当z轴向y轴转动时，角度为正值。
- roll：y轴和水平面的夹角，由于历史原因，范围为-90°至90°。
- 当x轴向z轴移动时，角度为正值。

传感器系统的使用

(4)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。
- 角加速度的单位是radians/second。
- 根据Nexus S手机实测：
  - 水平逆时针旋转，Z轴为正。
  - 水平逆时针旋转，z轴为负。
  - 向左旋转，y轴为负。
  - 向右旋转，y轴为正。
  - 向上旋转，x轴为负。
  - 向下旋转，x轴为正。

传感器系统的使用

(5)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。

传感器系统的使用

(6)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）。

(7) 温度传感器

- 温度传感器返回当前的温度。

传感器系统的使用

(8)接近(距离)传感器

- 接近传感器检测物体与手机的距离，单位是厘米。
- 一些接近传感器只能返回远和近两个状态，
- 因此，接近传感器将最大距离返回远状态，小于最大距离返回近状态。
- 接近传感器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

传感器系统的使用

- 从传感器管理器中获取某个或某些传感器的方法
  - 第 1 种：获取某种传感器的默认传感器

    ```
    Sensor defaultGyroscope =
    sensorManager.getDefaultSensor(Sensor.TYPE_GYROSCOPE);
    ```

  - 第 2 种：获取某种传感器的列表

    ```
    List<Sensor> pressureSensors =
    sensorManager.getSensorList(Sensor.TYPE_PRESSURE);
    ```

  - 第 3 种：获取所有传感器的列表

    ```
    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

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

- 例子（分3步）
  - 获得管理器.....获得传感器.....定义并注册事件监听器

```
//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);
```

```
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.
    }
};
```

学习目标：

- Android NDK
- Android传感器系统
- **位置服务**
- 地图应用

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
- 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