

每天都有免费的Android课堂学习



2015. Android学习

扣 2542--19189

学安卓就是这么简单

用代码谱写我们精彩人生!

Android 的传感器系统

Android 的传感器系统

- 第一部分 传感器系统综述
- 第二部分 传感器系统层次结构
- 第三部分 传感器系统的硬件抽象层
- 第四部分 传感器系统的使用

第一部分 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。

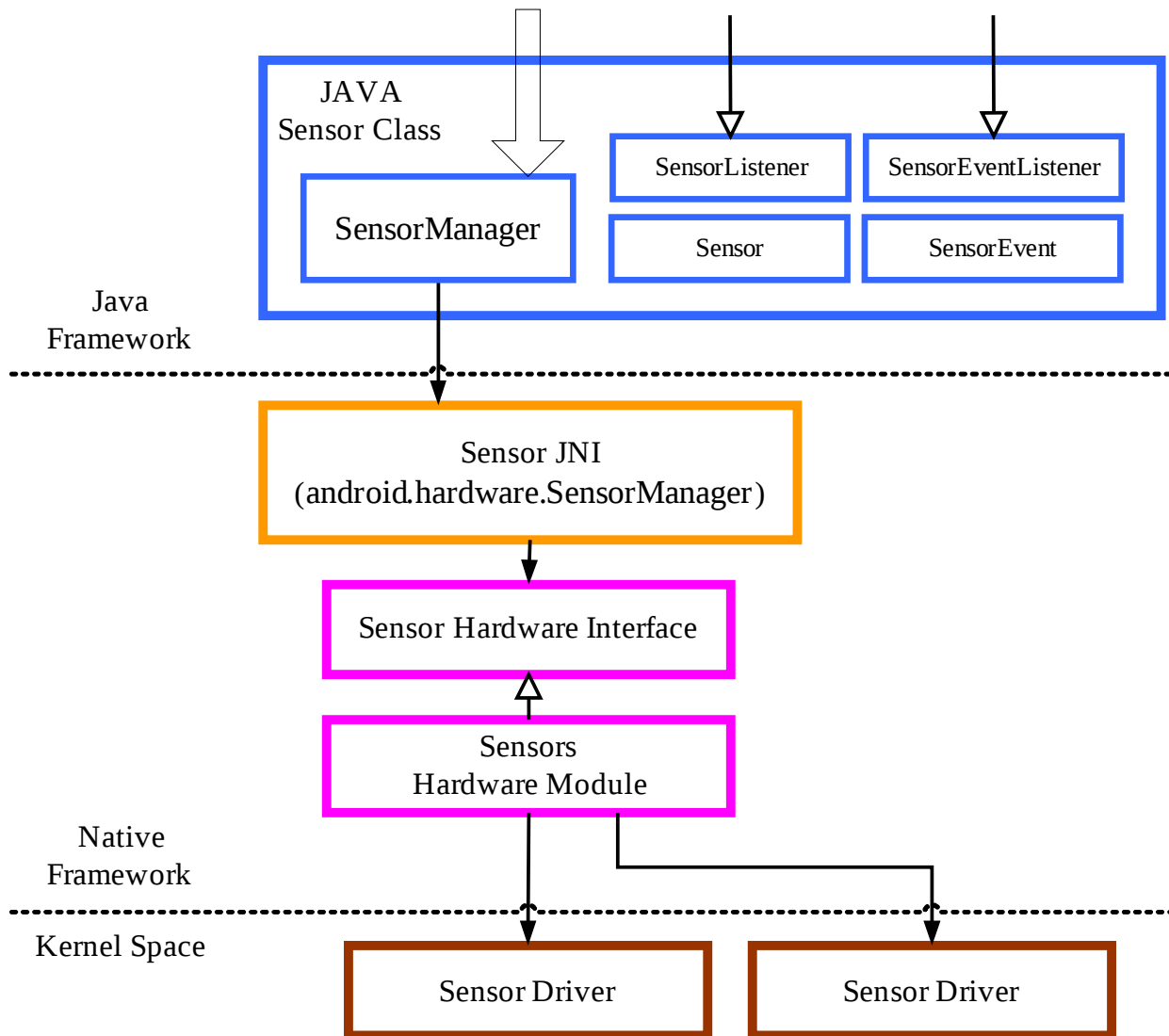
Android 的 **Sensor** 系统涉及了 **Android** 的各个层次。

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。

第一部分 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

第一部分 传感器系统综述



第一部分 传感器系统综述

Sensor 系统的代码分布情况如下所示:

1 . Sensor 的 JAVA 部分

代码路径: [frameworks/base/include/core/jave/android/hardware](#)

主要的代码为 Sensor*.java

2 . Sensor 的 JNI 部分

代码路径: [frameworks/base/core/jni](#)

android_hardware_SensorManager.cpp

3 . Sensor 硬件层实现的接口

头文件路径: [hardware/libhardware/include/hardware/sensors.h](#)

第二部分 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

第二部分 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 },
};
```

第二部分 传感器系统层次结构

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;
}
```

第二部分 传感器系统层次结构

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）

第二部分 传感器系统层次结构

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 }
}
```

第二部分 传感器系统层次结构

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。

第二部分 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);  
}
```

第三部分 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);
};
```

`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;
```

第三部分 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);  
}
```


第三部分 传感器的硬件抽象层

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;   /* 传感器的分辨率 */  
    float          power;        /* 传感器的耗能（估计值， mA 单位） */  
    void*          reserved[9];  
}
```

第三部分 Sensor 的硬件抽象层

Android 中为仿真器提供了一个为 Sensor 硬件抽象层的示例实现，它本身具有实际的功能，可以作为实际系统的传感器的硬件抽象层的示例。这部分代码如下所示：

[development/emulator/sensors](#)

第三部分 传感器系统的硬件抽象层

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。

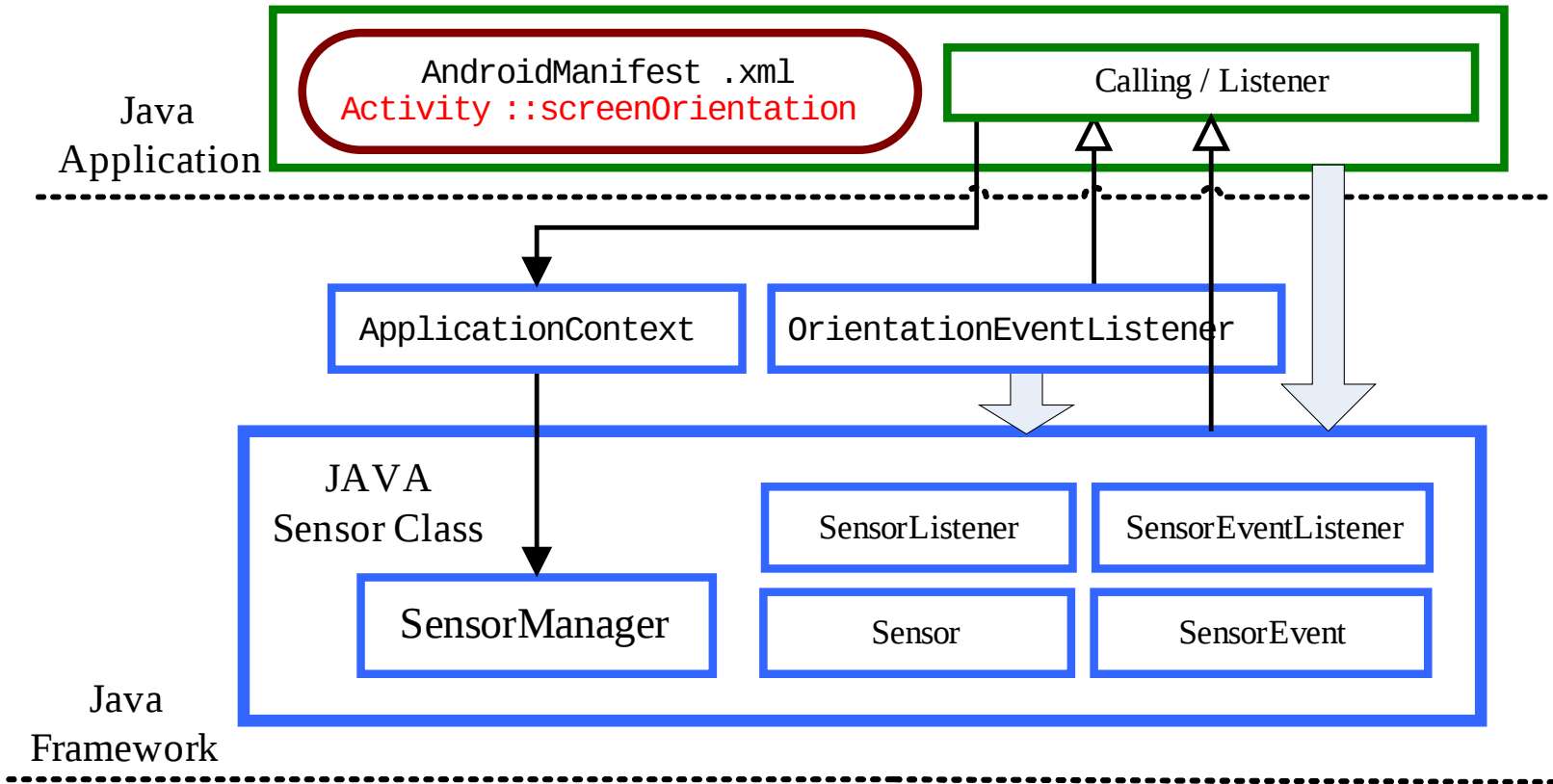
`sensors_data_t` 数据结构中的数值，是最终传感器传出的数据，在传感器的硬件抽象层中，需要构建这个数据。

第四部分 传感器系统的使用

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

第四部分 传感器系统的使用



谢谢！