



# *Android* 开发关键技术

华清远见上海 宋宝华

## 今天的内容

---

- } Android体系结构
- } Android内核
- } Android设备驱动
- } Android HAL结构
- } Android应用框架



# Android体系结构



# Android内核

---

- } Android对内核补丁
  - ∅ binder IPC系统
  - ∅ ashemem内存共享机制
  - ∅ Android Low Memory Killer
  - ∅ Android RAM console和log设备
  - ∅ Android alarm、timed\_gpio等
  - ∅ Android电源管理



# Android内核

---

- } Android对内核补丁
  - ∅ binder IPC系统
  - ∅ ashemem内存共享机制
  - ∅ Android Low Memory Killer
  - ∅ Android RAM console和log设备
  - ∅ Android alarm、timed\_gpio等
  - ∅ Android电源管理



# Android 设备驱动

---

- } 绝大部分为标准Linux驱动
- } 少数含自己的特点
  - o 对内核的补丁以驱动形式呈现
  - o 对USB、framebuffer、input的特殊要求



# Android USB驱动

Google没有使用原来的那套gadget驱动架构(file\_storage.c),而是参考file\_storage.c实现了一个新的模型-- composite模型:

```
composite.c           // 实现android下usb管理的框架模型
android.c             // 实现具体的usb功能管理
f_mass_storage.c     // 实现优盘功能
f_adb.c               // 实现adb功能
```

- 1)该框架下,有三个设备: composite设备, 优盘设备, adb设备
- 2)枚举时,首先枚举composite设备,再枚举优盘设备,最后枚举adb设备
- 3)composite设备被枚举时
  - a)在获取DEVICE描述符时,将VID,PID上报给host
  - b)host第一次请求CONFIG描述符时,composite设备告诉host,它有一个CONFIG,两个interface(即两个功能),以及告诉host自己使用的端点IN和端点OUT的地址
  - c)host会根据interface的个数决定枚举的次数(这对应着优盘枚举和adb枚举)

## Android input驱动(1)

keyboard驱动要通过input\_register\_device()调用注册成标准的输入设备/dev/input/event0

驱动上报的按键值要和Android系统里面的/system/usr/keylayout/\*.kl文件里面的记录一致,否则会导致android系统不能正确识别按键消息

```
key 232    DPAD_CENTER    WAKE_DROPPED
key 108    DPAD_DOWN      WAKE_DROPPED
key 103    DPAD_UP        WAKE_DROPPED
key 102    HOME           WAKE
key 105    DPAD_LEFT      WAKE_DROPPED
key 106    DPAD_RIGHT     WAKE_DROPPED
```

## Android input驱动(2)

---

Linux输入系统的任何一次事件通知包含如下三个信息:

事件类型 + 事件码 + 事件值(键值)

=>这3个信息的作用如下,举例说明:

对键盘输入设备来说 :事件类型EV\_KEY,表明是键盘送上来的数据  
事件码KEY\_CAMERA表明是键盘上的camera键被操作了  
事件值表示该camera键是按下还是松开了

对触摸屏输入设备来说 :事件类型EV\_ABS,表明是触摸屏送上来的绝对座标值数据  
事件码ABS\_X表明是触摸屏当前点的x座标  
事件值表示该当前点的x座标的具体绝对值

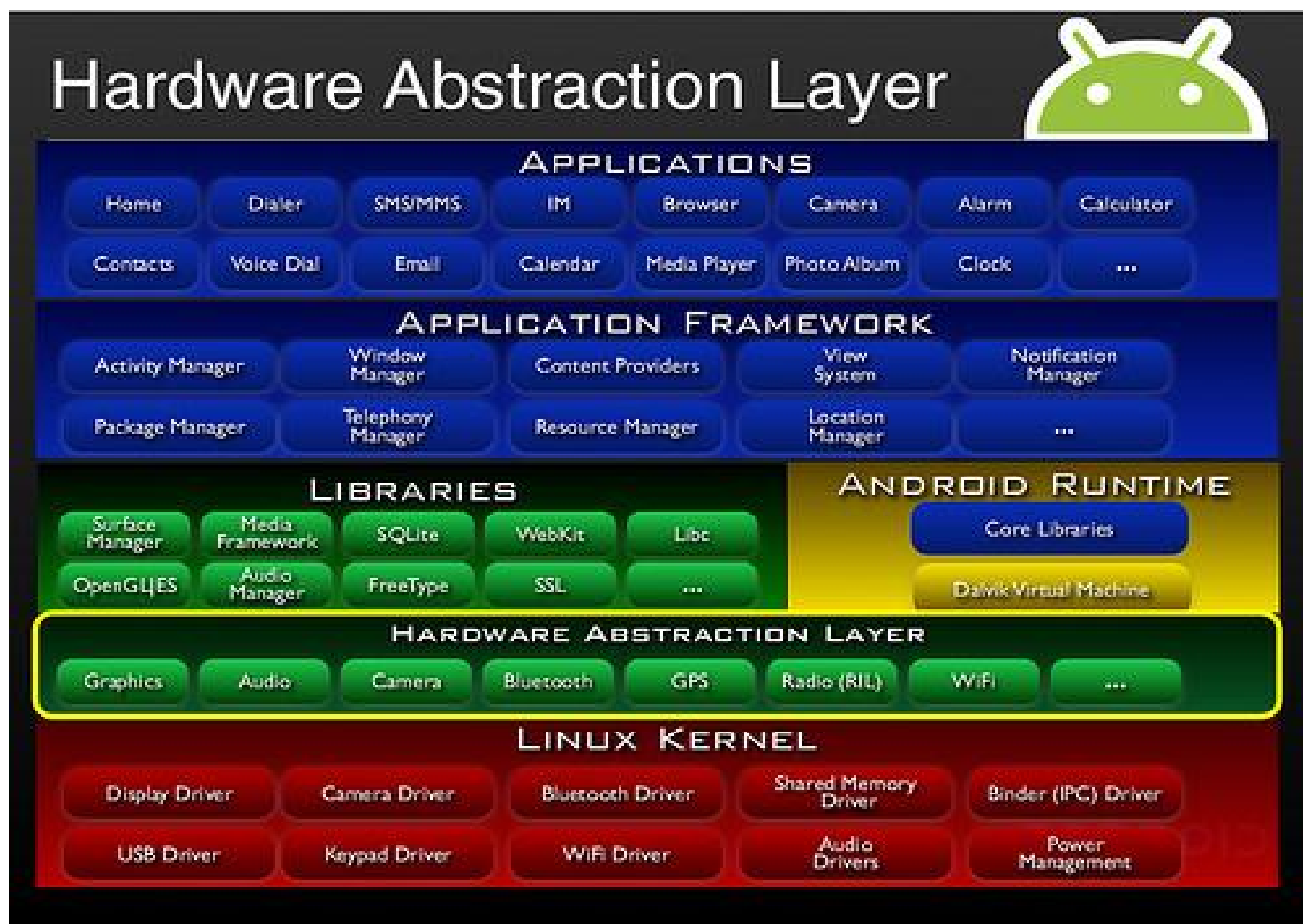
# Android framebuffer驱动

---

- } 要求支持double buffer
  - ∅  $\text{var} \rightarrow \text{yres\_virtual} = 2 * \text{var} \rightarrow \text{yres}$
- } 支持double buffer之间的切换
  - ∅ 添加pan函数

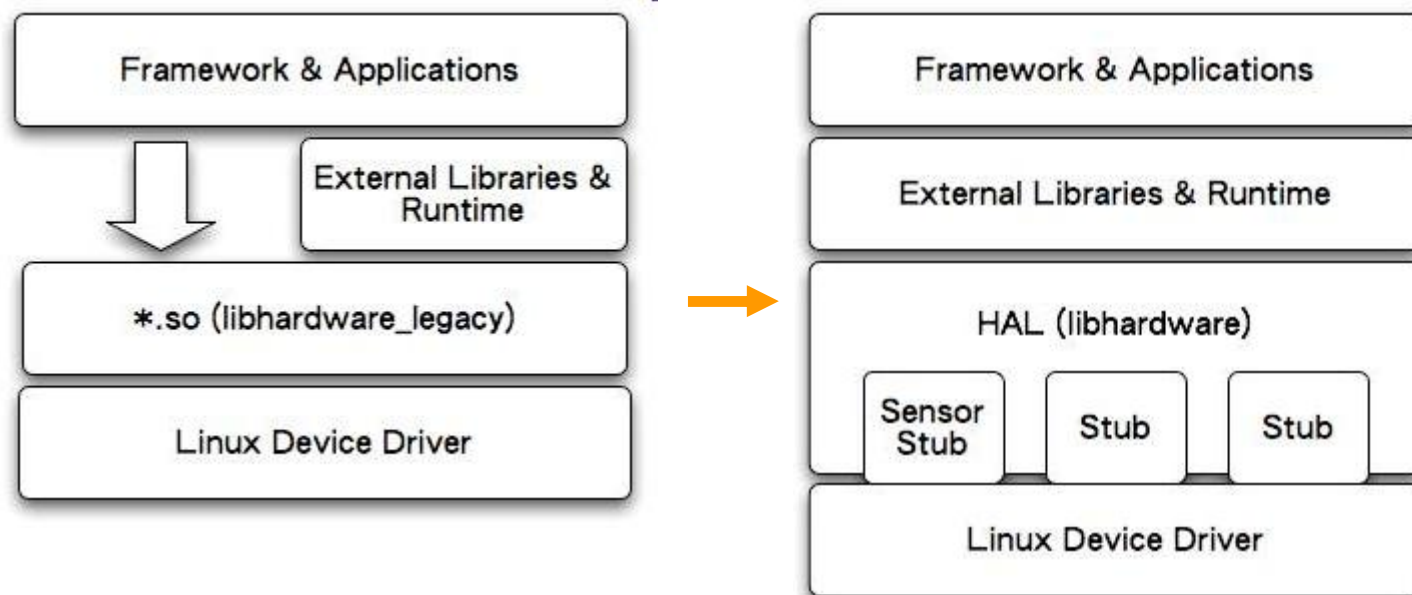


# Android HAL结构(1)



## Android HAL结构(2)

- } HAL 的目的是为了把 Android framework 与 Linux kernel 完整「隔开」



# Android HAL源代码位置



- } 在 Android源代码里，HAL 主要的实作储存于以下目录：
- Ø 1. libhardware\_legacy/ - 旧的实例、采取链接库模块的形式进行；直接函数调用
  - Ø 2. libhardware/ - 新版的实例、调整为 HAL stub 的形式；间接函数调用
  - Ø 3. ril/ - Radio Interface Layer



# Android HAL模块代码结构与用法

## } HAL模块代码结构:

- Ø struct hw\_module\_t;
- Ø struct hw\_module\_methods\_t;
- Ø struct hw\_device\_t;

## } HAL模块的用法:

- Ø Native code通过hw\_get\_module调用获取HAL stub: hw\_get\_module(LED\_HARDWARE\_MODULE\_ID, (const hw\_module\_t\*\*) &module)
- Ø 通过继承hw\_module\_methods\_t的callback来open设备:  
module->methods->open(module,  
LED\_HARDWARE\_MODULE\_ID, (struct hw\_device\_t\*\*) device);
- Ø 通过继承hw\_device\_t的callback来控制设备:  
sLedDevice->set\_on(sLedDevice, led);  
sLedDevice->set\_off(sLedDevice, led);

# Android 应用编程

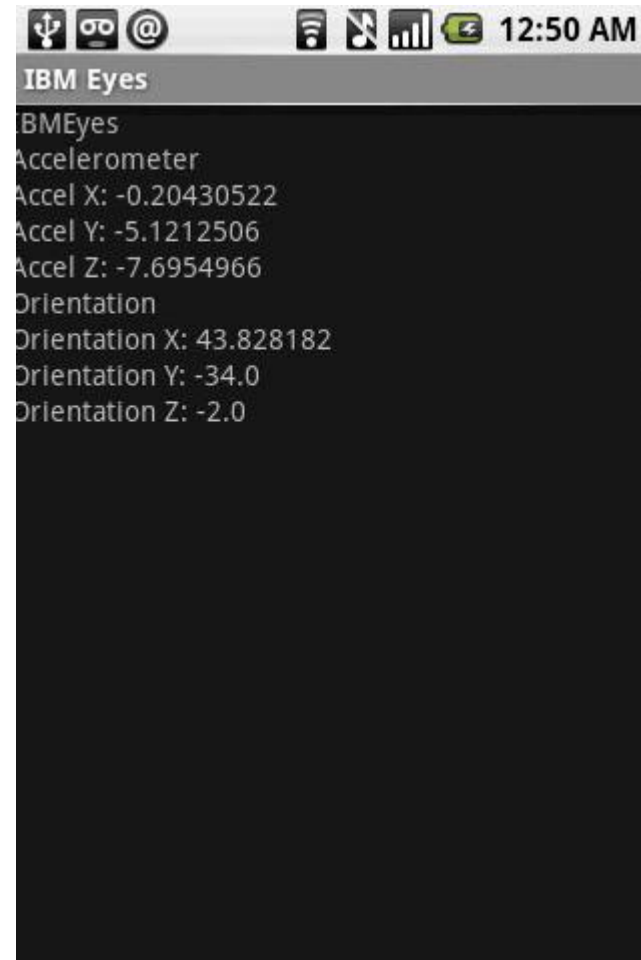
## } 应用程序使用与具体硬件无关的方式访问HAL

```
private SensorEventListener acc_listener = new SensorEventListener() {
    public void onAccuracyChanged(Sensor sensor, int accuracy) {
        // TODO Auto-generated method stub
    }

    public void onSensorChanged(SensorEvent event) {
        // TODO Auto-generated method stub
        Log.i("-----", "" + "TYPE_ACCELEROMETER");
        xViewA.setText("ACCELEROMETER_X: " + event.values[0]);
        yViewA.setText("ACCELEROMETER_Y: " + event.values[1]);
        zViewA.setText("ACCELEROMETER_Z: " + event.values[2]);
    }
};

public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    sm = (SensorManager) getSystemService(SENSOR_SERVICE);
    setContentView(R.layout.main);
    xViewA = (TextView) findViewById(R.id.xbox);
    yViewA = (TextView) findViewById(R.id.ybox);
    zViewA = (TextView) findViewById(R.id.zbox);

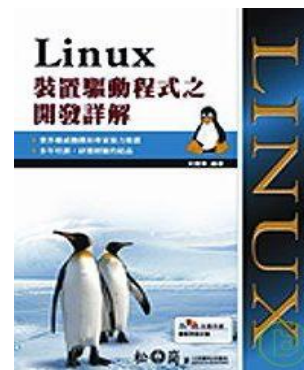
    sm.registerListener(acc_listener, sm.getDefaultSensor(Sensor.TYPE_ACCELEROMETER),
        SensorManager.SENSOR_DELAY_NORMAL);
    sm.registerListener(ori_listener, sm.getDefaultSensor(Sensor.TYPE_ORIENTATION),
        SensorManager.SENSOR_DELAY_NORMAL);
    sm.registerListener(mag_listener, sm.getDefaultSensor(Sensor.TYPE_MAGNETIC_FIELD),
        SensorManager.SENSOR_DELAY_NORMAL);
}
```



# Linux设备驱动开发详解

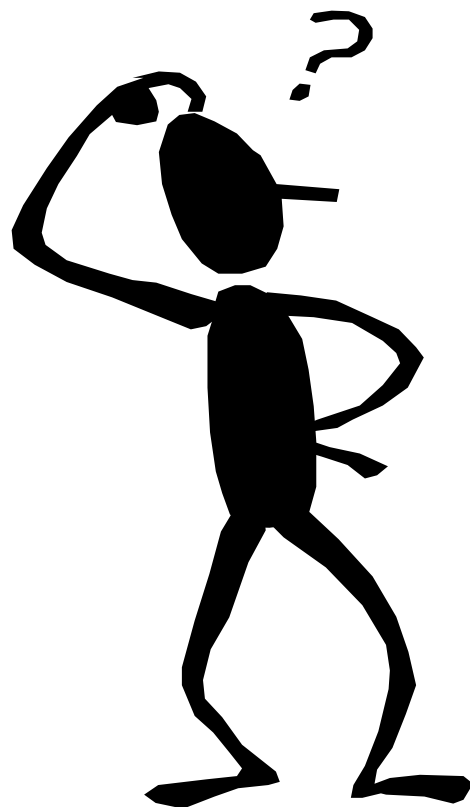
## } 主要出发点:

- 力求用最简单的实例讲解复杂的知识点，以免实例太复杂搅浑读者（驱动理论部分）
- 对Linux设备驱动多种复杂设备的框架结构进行了全面的介绍（驱动框架部分）
- 更面向实际的嵌入式工程，讲解开发必备的软硬件基础，及开发手段（调试与移植部分）
- 提供讨论与交流平台（华清远见，[www.linuxdriver.cn](http://www.linuxdriver.cn)）



# 让我们一起讨论！

---



谢谢!

