Android 系统非标准设备驱动程序设计

孟小华，黄宗轩
(暨南大学 计算机科学系，广东 广州 510632)

摘 要：在深入研究 Android 硬件抽象层 HAL 和 Java 本地接口 JNI 技术原理的基础上，提出了一个 Android 非标准硬件驱动程序的设计方案。以一个非标准设备的驱动程式的实现为例介绍了驱动程序的功能模块分层设计，讨论了使用 HAL Stub 技术对硬件抽象层 HAL 模块进行优化的方法。

关键词：Android；设备驱动程序；硬件抽象层；JNI
中图分类号：TP316.8 文献标识码：A 文章编号：1674-7720(2011)14-0007-03

The design of non-standard device driver for Android

Meng Xiaohua, Huang Zongxuan
(Department of Computer Science, Jinan University, Guangzhou, Guangdong 510632, China)

Abstract: This paper gives a solution of non-standard device driver design on the basis of in-depth analyzing the principle of HAL and JNI. It introduces the layering design of the function module of driver through an implementation of device driver as example. Then it discusses the driver's optimization methods of the hardware abstraction layer by using HAL stub.

Key words: Android; device driver; HAL; JNI

Android 系统是 Google 推出的基于 Linux 内核和 Java 架构的操作系统，在很短的时间内已成为主流的手机操作系统，并已逐步扩展应用到嵌入式系统、平板电脑和上网本上。它既有 Linux 系统所具有的硬件平台可移植性，也因使用 Java 语言开发应用程序带来了应用软件只编写一次即可在所有平台运行的巨大优势。Android 虽然主要基于已有的技术，但在体系结构设计上有较大的创新。其主要设计目标之一就是要使应用程序和系统能独立于具体的计算机体系结构和硬件平台，表现在设备驱动程序设计上，对于已有的 Linux 标准设备驱动程序可以直接继续使用，只需为其增加应用层 JNI 接口。但对于 Linux 没有的非标准设备则提倡在 Linux 内核中驱动部分只做很少的接口工作，尽量把驱动程序的主要处理放在 Android 的上层架构中，即在应用层实现。本文对 Android 系统的底层实现技术进行深入的研究，包括 Android 的硬件抽象层和 JNI 技术实现等。并以 S3C2440 开发板上的 LED 灯设计显示驱动程序为例，提出了一种非标准硬件设备驱动程序的设计和实现方案。

1 Android 系统驱动程序架构

1.1 驱动程序分层体系结构

Android 是基于 Linux 的，它使用了 Linux 内核，但应用程序使用 Java 语言开发，所以应用程序在调用设备驱动时不能像一般的 Linux 应用程序那样直接使用系统调用，必须通过 Java 虚拟机的 JNI 的本地(Native)方法使用设备。另一方面，Android 要成为一个通用性强的平台，必须加强它的可移植性。这也是在 Android 架构添加一个硬件抽象层(HAL)的原因，目的是为设备的调用提供一个更高级的封装图 1 所示为 Android 驱动程序架构。

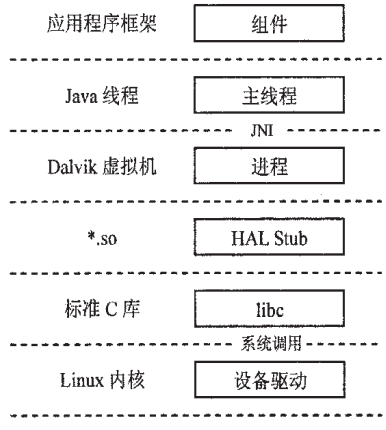

图 1 Android 驱动程序架构

HAL Stub 是以 Linux 共享库(*.so)的形式存在,在整个驱动架构中,它是设备驱动程序运行在用户空间的一部分,它向上为 Dalvik 虚拟机提供硬件设备的抽象接口,向下通过系统调用与 Linux 内核中的驱动程序进行数据交互。在这个过程中 HAL 可以对驱动程序的数据进行处理,也就是说在 Linux 内核中的驱动程序部分只需要提供一个与硬件设备传输数据接口的功能,而其余具体的操作可以由 HAL 完成。

1.2 Android 的硬件抽象层

Android 的硬件抽象层 HAL(Hardware Abstract Layer)在 Android 的架构中是在库这一层中,通过这一层,硬件厂商可以把部分设备的驱动源码封装在这一层而不公开源代码。

对图 1 分析,设计 HAL 就是为了把应用框架和 Linux 内核分离出来,让 Android 使用 Linux 内核而又不完全依赖 Linux 内核。当然,驱动程序并不是完全从 Linux 内核中分离出来,一些基本的处理必须由内核来完成,HAL 只是分担了 Linux 设备驱动的部分功能,至于这部分的功能占驱动程序功能的比例目前并没有一个标准。

在 Android 系统发展过程中,HAL 的实现也逐步有了一些变化,旧的 HAL 是一种模块化的思想,通过共享库的形式由 Runtime 在 JNI 时以函数调用方法调用,这种做法并没有通过封装,即上层应用可以直接调用硬件。另外,这种方法可被多个进程使用,映射到多个进程空间中浪费内存资源。

现在 HAL 提出一种 Stub 的思想,HAL Stub 是一种代理的概念,Stub 同样是以共享库(*.so)格式存在,但上层应用并不像加载动态库那样调用 Stub。这种 HAL 是由模块与 Stub 结合而成,Runtime 通过模块提供的统一接口获取并操作 Stub。Stub 向 HAL 提供操作的回调函数,Runtime 向 HAL 取得指定模块的操作函数后,调用这些回调函数。这是一种间接函数调用的方式,HAL 里包含了多个 Stub。图 2 为 HAL Stib 原理。

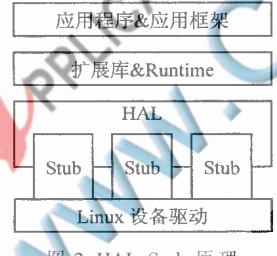

图 2 HAL Stub 原理

1.3 Android 的 JNI 实现原理

JNI 是 Java Native Interface 的缩写,是在 Sun 的 Java 平台中首先定义出来的,它允许 Java 代码与其他语言代码进行交互。Android 中 JNI 的设计目的也是一样:

(1) 应用程序需要与硬件平台交互时,Java 库中的类不可能支持;

(2) 本地已经使用其他语言编写的库允许 Java 程序访问;

(3) 某些功能用较低级的语言实现的执行效率较高,让 Java 程序调用这些函数。

在 Android 应用层中的程序或组件都是用 Java 语言开发的,这些 Java 代码编译后变成 Dex 格式的字节码,由 Dalvik 虚拟机执行,在执行过程中需要调用本地库时,由虚拟机载入这些本地库,然后让 Java 函数调用库中的函数,虚拟机相当于一座桥梁,让 Java 与本地库能够透过标准的 JNI 界面互相沟通。

应用程序在虚拟机里执行,通过函数 System.loadLibrary( )通知虚拟机载入指定的库,例如在 Java 代码中包含代码如:

```
… …
System.loadLibrary("sample_jni");
… …
```

虚拟机就会在 Android 文件系统的"/system/lib/"目录中查找 libsample_jni.so 库文件,虚拟机载入 libsample_jni.so 后,Java 代码就可以与库文件结合起来一起执行。

这些用 C 语言编写的本地库必须遵循规范,当虚拟机执行 System.loadLibrary()函数时,首先执行本地库里的 JNI_OnLoad()函数,这个函数需要实现的功能是:返回给虚拟机此本地库使用的 JNI 版本;对库进行初始化。如果本地库里没有实现 JNI_OnLoad()函数,虚拟机就会默认本地库使用最老的 JNI 1.1 版本。

JNI_OnUnload()函数与装入函数相对应,在虚拟机释放该本地库时,会调用 JNI_OnUnload()函数进行资源回收动作。

在应用层的 Java 代码通过虚拟机调用本地函数,一般要依赖于虚拟机查找库里的本地函数,如果需要调用比较频繁,每次都要寻找一遍,就会花费较多的时间影响效率,在这里可以通过 registerNativeMethods()函数把 gMethods[]表格所含的本地函数注册到虚拟机里。

2 Android 硬件驱动程序设计

Android 是一个开放平台,在嵌入式移动设备领域里具有很好的应用前景,但在不同的设备上往往有不同的硬件支持,要在 Android 中添加这些硬件应用,不是单纯地在 Linux 内核中添加驱动模块,还必须在用户空间和应用框架中添加对应的支持。下面以给 S3C2440 开发板添加一个 LED 显示控制驱动功能为例展示 Android 平台添加新硬件支持的过程。

2.1 硬件驱动程序的框架

LED 控制功能通过应用程序来开关开发板上的 LED 灯。在应用层中 LED 控制程序调用 LED 控制服务(Android Service),应用层中的 LED 控制服务通过 JNI 让虚拟机加载 LED 控制的本地库,然后向 HAL 获取 LED Stub,由 Stub 调用在 Linux 内核中的 LED 驱动。图 3 为 LED 控制功能的架构设计。

从 LED 控制功能的架构来分,整个功能可以分成五个模块:LED 驱动模块、LED Stub 模块、LED 本地服务模块、LED 服务管理模块和 LED 应用模块。

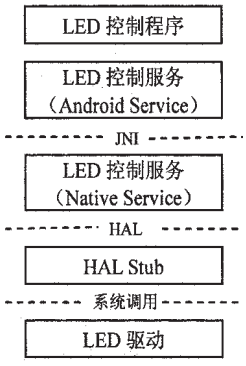

图 3 LED 控制功能的架构设计

2.2 HAL 中的 Stub 的设计与实现

图 4 是 LED Stub 的实现过程。LED Stub 是硬件抽象层中 LED 控制的代理，当 LED 控制的本地服务需要调用 LED Stub 时，通过函数 hw_get_module( ) 结合 LED Stub 的模块 ID 向 HAL 申请 LED Stub，本地服务获得 Stub 对象后，可以把 Stub 看作一个抽象硬件进行操作。

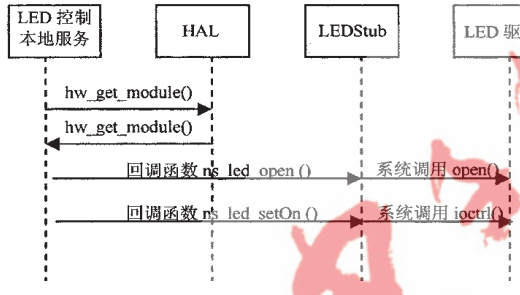

图 4 LED Stub 的实现过程

下面是定义 LED Stub 的 HAL 结构体：

```
struct led_module_t {
    struct hw_module_t common;
}
struct led_module_t {
    struct hw_module_t common;
    int fd;
    int(*ns_set_on)(struct led_control_device_t*dev,int32_t led);
    int(*ns_set_off)(struct led_control_device_t*dev,int32_t led);
}
```

将结构体 led_module_t 初始化一个实例名为 HAL_MODULE_INFO_SYM，这个名称不能修改，实例里包含了 Stub 的模块信息，主要包括：

tag：标记了结构体的类型，这里的值为 HARDWARE_MODULE_TAG；

id：LED Stub 的模块 ID，在本地服务向 HAL 获取 Stub 时调用的函数 hw_get_module()中，通过这里的 id 查找 LED Stub；

methods：是结构体 hw_module_methods_t 的实例，为 HAL 定义回调函数 open()。

这里的 open () 函数是一个必须实现的回调函数接口，在本地服务获得 Stub 对象后调用，它负责申请结构体 led_control_device_t 的空间，填充信息，注册具体操作的回调函数接口并打开 LED 驱动。

结构体 led_control_device_t 继承了 hw_device_t，在 open()函数调用时填充的主要信息包括：

tag：结构体的类型，这里的值为 HARDWARE_DEVICE_TAG；

module：Stub 的模块，也就是实例 HAL_MODULE_INFO_SYM 中的 hw_module_t 部分；

close：释放 LED Stub 的回调函数；

fd：打开设备驱动文件返回的文件描述符；

ns_set_on：打开 LED 灯的回调函数指针；

ns_set_off：关闭 LED 灯的回调函数指针。

回调函数指针“*ns_set_on”和“*ns_set_off”分别指向实现函数 hal_led_on()和 hal_led_off()，在实现函数中通过系统调用 ioctl()对 LED 灯进行开关控制。

2.3 硬件控制服务的 JNI 实现

LED 控制本地库编译后为“libled.so”保存在 Android 文件系统的“/sysem/lib/”目录下面，LED 控制服务的 Android 进程运行后由虚拟机实例装入本地库，具体实现过程如图 5 所示。

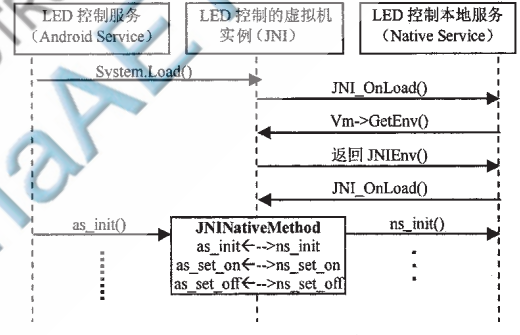

图 5 LED 控制服务的 JNI 实现过程

LED 控制服务调用 System.load()函数，它的虚拟机实例就会装入 LED 控制本地库，虚拟机会首先调用 JNI_OnLoad()函数完成：

(1) 把虚拟机环境信息保存到本地库的一个结构体“JNIEnv”的实例中；

(2) 建立一个应用层中的 LED 控制服务与本地库的 JNI 函数表；

(3) 返回虚拟机本地库使用的 JNI 版本。

加载完后，应用层中的 LED 控制服务就可以通过虚拟机中的 JNI 函数表把运行的 Java 函数转换为本地函数执行。在 LED 控制服务类中定义有 JNI 函数的方式，例如下面的代码段：

```
public final class LedService extends IledService.Stub {
    ……
    static {
        System.load(“/system/lib/libled.so”);
```

```
    }
    ……
    private static native boolean as_init();
    private static native boolean as_set_on(int led);
    private static native boolean as_set_off(int led);
}
```

本文的研究工作是在S3C2440开发板上进行的，以给开发板上的LED灯增加驱动程序为例，展示了一种为Android平台非标准硬件增加驱动程序的设计方案，对于实现其他设备的驱动具有一定的借鉴意义。由于各种硬件设备及其接口差异较大，本文着重于驱动程序的设计方案，没有讨论相关的硬件接口驱动细节。随着Android平台日渐成熟以及应用数量的增加，它在嵌入式领域的应用范围将会更加广泛。为Android设备编写不同于标准Linux系统的设备驱动程序会变得越来越多。

参考文献

[1] BRADY P. Anatomy & physiology of an android[EB/OL]. 2008[2009-03-24].http://sites.google.com/site/io/anatomy-physiology-of-an-android.

[2] GREG K H. Android and the linux kernel community[EB/OL].[2010-02-02], http://www.kroah.com/log/linux/android-kernel-problems.html.

[3] 叶炳发, 孟小华. Android图形系统的分析与移植[J]. 电信科学,2010(02):65-68.

[4] 李俊.嵌入式Linux设备驱动程序开发详解[M]. 北京：人民邮电出版社,2008.

[5] 胡希明,毛德操. Linux内核源代码情景分析[M]. 杭州：浙江大学出版社,2004.

[6] 姚昱旻, 刘卫国. Android的架构与应用开发研究 [D]. 长沙:中南大学,2008.

(收稿日期:2011-03-21)

作者简介：

孟小华，男，1965年生，副教授，硕士生导师，主要研究方向：嵌入式系统，计算机网络。

黄宗轩，男，1985年生，硕士生，主要研究方向：计算机应用。