在Ubuntu上为Android增加硬件抽象层（HAL）模块访问Linux内核驱动程序

在Android硬件抽象层（HAL）概要介绍和学习计划一文中，我们简要介绍了在Android系统为为硬件编写驱动程序的方法。简单来说，硬件驱动程序一方面分布在Linux内核中，另一方面分布在用户空间的硬件抽象层中。接着，在Ubuntu上为Android系统编写Linux内核驱动程序一文中举例子说明了如何在Linux内核编写驱动程序。在这一篇文章中，我们将继续介绍Android系统硬件驱动程序的另一方面实现，即如何在硬件抽象层中增加硬件模块来和内核驱动程序交互。在这篇文章中，我们还将学习到如何在Android系统创建设备文件时用类似Linux的udev规则修改设备文件模式的方法。

一. 参照在Ubuntu上为Android系统编写Linux内核驱动程序一文所示，准备好示例内核驱动序。完成这个内核驱动程序后，便可以在Android系统中得到三个文件，分别是/dev/hello、/sys/class/hello/hello/val和/proc/hello。在本文中，我们将通过设备文件/dev/hello来连接硬件抽象层模块和Linux内核驱动程序模块。

二. 进入到在hardware/libhardware/include/hardware目录，新建hello.h文件：

USER-NAME@MACHINE-NAME:~/Android$ cd hardware/libhardware/include/hardware

USER-NAME@MACHINE-NAME:~/Android/hardware/libhardware/include/hardware$ vi hello.h

hello.h文件的内容如下：

```cpp
#ifndef ANDROID_HELLO_INTERFACE_H
#define ANDROID_HELLO_INTERFACE_H
#include <hardware/hardware.h>

__BEGIN_DECLS

/*定义模块ID*/
#define HELLO_HARDWARE_MODULE_ID "hello"

/*硬件模块结构体*/
struct hello_module_t {
    struct hw_module_t common;
};

/*硬件接口结构体*/
struct hello_device_t {
    struct hw_device_t common;
    int fd;
    int (*set_val)(struct hello_device_t* dev, int val);
    int (*get_val)(struct hello_device_t* dev, int* val);
};

__END_DECLS

#endif
```

这里按照Android硬件抽象层规范的要求，分别定义模块ID、模块结构体以及硬件接口结构

体。在硬件接口结构体中，fd表示设备文件描述符，对应我们将要处理的设备文件"/dev/hello"，set_val和get_val为该HAL对上提供的函数接口。

三. 进入到hardware/libhardware/modules目录，新建hello目录，并添加hello.c文件。hello.c的内容较多，我们分段来看。

首先是包含相关头文件和定义相关结构：

```cpp
#define LOG_TAG "HelloStub"

#include <hardware/hardware.h>
#include <hardware/hello.h>
#include <fcntl.h>
#include <errno.h>
#include <cutils/log.h>
#include <cutils/atomic.h>

#define DEVICE_NAME "/dev/hello"
#define MODULE_NAME "Hello"
#define MODULE_AUTHOR "shyluo@gmail.com"

/*设备打开和关闭接口*/
static int hello_device_open(const struct hw_module_t* module, const char* name, struct
static int hello_device_close(struct hw_device_t* device);

/*设备访问接口*/
static int hello_set_val(struct hello_device_t* dev, int val);
static int hello_get_val(struct hello_device_t* dev, int* val);

/*模块方法表*/
static struct hw_module_methods_t hello_module_methods = {
    open: hello_device_open
};

/*模块实例变量*/
struct hello_module_t HAL_MODULE_INFO_SYM = {
    common: {
        tag: HARDWARE_MODULE_TAG,
        version_major: 1,
        version_minor: 0,
        id: HELLO_HARDWARE_MODULE_ID,
        name: MODULE_NAME,
        author: MODULE_AUTHOR,
        methods: &hello_module_methods,
    }
};
```

这里，实例变量名必须为HAL_MODULE_INFO_SYM，tag也必须为HARDWARE_MODULE_TAG，这是Android硬件抽象层规范规定的。

定义hello_device_open函数：

```cpp
static int hello_device_open(const struct hw_module_t* module, const char* name, struct
    struct hello_device_t* dev;dev = (struct hello_device_t*)malloc(sizeof(struct hello

    if(!dev) {
        LOGE("Hello Stub: failed to alloc space");
        return -EFAULT;
    }

    memset(dev, 0, sizeof(struct hello_device_t));
    dev->common.tag = HARDWARE_DEVICE_TAG;
    dev->common.version = 0;
    dev->common.module = (hw_module_t*)module;
    dev->common.close = hello_device_close;
    dev->set_val = hello_set_val;dev->get_val = hello_get_val;

    if((dev->fd = open(DEVICE_NAME, O_RDWR)) == -1) {
        LOGE("Hello Stub: failed to open /dev/hello -
- %s.", strerror(errno));free(dev);
        return -EFAULT;
    }

    *device = &(dev->common);
    LOGI("Hello Stub: open /dev/hello successfully.");

    return 0;
}
```

DEVICE_NAME定义为"/dev/hello"。由于设备文件是在内核驱动里面通过device_create创建的，而device_create创建的设备文件默认只有root用户可读写，而hello_device_open一般是由上层APP来调用的，这些APP一般不具有root权限，这时候就导致打开设备文件失败：

Hello Stub: failed to open /dev/hello -- Permission denied.

解决办法是类似于Linux的udev规则，打开Android源代码工程目录下，进入到system/core/rootdir目录，里面有一个名为ueventd.rc文件，往里面添加一行：

/dev/hello 0666 root root

定义hello_device_close、hello_set_val和hello_get_val这三个函数：

```cpp
static int hello_device_close(struct hw_device_t* device) {
    struct hello_device_t* hello_device = (struct hello_device_t*)device;

    if(hello_device) {
        close(hello_device->fd);
        free(hello_device);
    }
```

```
08.
09.     return 0;
10. }
11.
12. static int hello_set_val(struct hello_device_t* dev, int val) {
13.     LOGI("Hello Stub: set value %d to device.", val);
14.
15.     write(dev->fd, &val, sizeof(val));
16.
17.     return 0;
18. }
19.
20. static int hello_get_val(struct hello_device_t* dev, int* val) {
21.     if(!val) {
22.         LOGE("Hello Stub: error val pointer");
23.         return -EFAULT;
24.     }
25.
26.     read(dev->fd, val, sizeof(*val));
27.
28.     LOGI("Hello Stub: get value %d from device", *val);
29.
30.     return 0;
31. }
```

四. 继续在hello目录下新建Android.mk文件：

LOCAL_PATH := $(call my-dir)
include $(CLEAR_VARS)
LOCAL_MODULE_TAGS := optional
LOCAL_PRELINK_MODULE := false
LOCAL_MODULE_PATH := $(TARGET_OUT_SHARED_LIBRARIES)/hw
LOCAL_SHARED_LIBRARIES := liblog
LOCAL_SRC_FILES := hello.c
LOCAL_MODULE := hello.default
include $(BUILD_SHARED_LIBRARY)

注意，LOCAL_MODULE的定义规则，hello后面跟有default，hello.default能够保证我们的模块总能被硬象抽象层加载到。

五. 编译：

USER-NAME@MACHINE-NAME:~/Android$ mmm hardware/libhardware/modules/hello

编译成功后，就可以在out/target/product/generic/system/lib/hw目录下看到hello.default.so文件了。

六. 重新打包Android系统镜像system.img：

USER-NAME@MACHINE-NAME:~/Android$ make snod

重新打包后，system.img就包含我们定义的硬件抽象层模块hello.default了。

虽然我们在Android系统为我们自己的硬件增加了一个硬件抽象层模块，但是现在Java应用程序还不能访问到我们的硬件。我们还必须编写JNI方法和在Android的Application Frameworks层增加API接口，才能让上层Application访问我们的硬件。在接下来的文章中，我

们还将完成这一系统过程，使得我们能够在Java应用程序中访问我们自己定制的硬件。