# 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](http://blog.csdn.net/luoshengyang/article/details/6567257" \t "_blank)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](http://blog.csdn.net/luoshengyang/article/details/6568411" \t "_blank)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。

**《Android系统源代码情景分析》一书正在进击的程序员网（**[http://0xcc0xcd.com](http://0xcc0xcd.com/" \t "_blank)**）中连载，点击进入！**

      一. 参照[在Ubuntu上为Android系统编写Linux内核驱动程序](http://blog.csdn.net/luoshengyang/article/details/6568411" \t "_blank)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]** [view plain](https://blog.csdn.net/luoshengyang/article/details/6573809) [copy](https://blog.csdn.net/luoshengyang/article/details/6573809)

1. #ifndef ANDROID\_HELLO\_INTERFACE\_H
2. #define ANDROID\_HELLO\_INTERFACE\_H
3. #include <hardware/hardware.h>
5. \_\_BEGIN\_DECLS
7. /\*定义模块ID\*/
8. #define HELLO\_HARDWARE\_MODULE\_ID "hello"
10. /\*硬件模块结构体\*/
11. **struct** hello\_module\_t {
12. **struct** hw\_module\_t common;
13. };
15. /\*硬件接口结构体\*/
16. **struct** hello\_device\_t {
17. **struct** hw\_device\_t common;
18. **int** fd;
19. **int** (\*set\_val)(**struct** hello\_device\_t\* dev, **int** val);
20. **int** (\*get\_val)(**struct** hello\_device\_t\* dev, **int**\* val);
21. };
23. \_\_END\_DECLS
25. #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]** [view plain](https://blog.csdn.net/luoshengyang/article/details/6573809) [copy](https://blog.csdn.net/luoshengyang/article/details/6573809)

1. #define LOG\_TAG "HelloStub"
3. #include <hardware/hardware.h>
4. #include <hardware/hello.h>
5. #include <fcntl.h>
6. #include <errno.h>
7. #include <cutils/log.h>
8. #include <cutils/atomic.h>
10. #define DEVICE\_NAME "/dev/hello"
11. #define MODULE\_NAME "Hello"
12. #define MODULE\_AUTHOR "shyluo@gmail.com"
14. /\*设备打开和关闭接口\*/
15. **static** **int** hello\_device\_open(**const** **struct** hw\_module\_t\* module, **const** **char**\* name, **struct** hw\_device\_t\*\* device);
16. **static** **int** hello\_device\_close(**struct** hw\_device\_t\* device);
18. /\*设备访问接口\*/
19. **static** **int** hello\_set\_val(**struct** hello\_device\_t\* dev, **int** val);
20. **static** **int** hello\_get\_val(**struct** hello\_device\_t\* dev, **int**\* val);
22. /\*模块方法表\*/
23. **static** **struct** hw\_module\_methods\_t hello\_module\_methods = {
24. open: hello\_device\_open
25. };
27. /\*模块实例变量\*/
28. **struct** hello\_module\_t HAL\_MODULE\_INFO\_SYM = {
29. common: {
30. tag: HARDWARE\_MODULE\_TAG,
31. version\_major: 1,
32. version\_minor: 0,
33. id: HELLO\_HARDWARE\_MODULE\_ID,
34. name: MODULE\_NAME,
35. author: MODULE\_AUTHOR,
36. methods: &hello\_module\_methods,
37. }
38. };

      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]** [view plain](https://blog.csdn.net/luoshengyang/article/details/6573809) [copy](https://blog.csdn.net/luoshengyang/article/details/6573809)

1. **static** **int** hello\_device\_open(**const** **struct** hw\_module\_t\* module, **const** **char**\* name, **struct** hw\_device\_t\*\* device) {
2. **struct** hello\_device\_t\* dev;dev = (**struct** hello\_device\_t\*)malloc(**sizeof**(**struct** hello\_device\_t));
4. **if**(!dev) {
5. LOGE("Hello Stub: failed to alloc space");
6. **return** -EFAULT;
7. }
9. memset(dev, 0, **sizeof**(**struct** hello\_device\_t));
10. dev->common.tag = HARDWARE\_DEVICE\_TAG;
11. dev->common.version = 0;
12. dev->common.module = (hw\_module\_t\*)module;
13. dev->common.close = hello\_device\_close;
14. dev->set\_val = hello\_set\_val;dev->get\_val = hello\_get\_val;
16. **if**((dev->fd = open(DEVICE\_NAME, O\_RDWR)) == -1) {
17. LOGE("Hello Stub: failed to open /dev/hello -- %s.", strerror(errno));free(dev);
18. **return** -EFAULT;
19. }
21. \*device = &(dev->common);
22. LOGI("Hello Stub: open /dev/hello successfully.");
24. **return** 0;
25. }

      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]** [view plain](https://blog.csdn.net/luoshengyang/article/details/6573809) [copy](https://blog.csdn.net/luoshengyang/article/details/6573809)

1. **static** **int** hello\_device\_close(**struct** hw\_device\_t\* device) {
2. **struct** hello\_device\_t\* hello\_device = (**struct** hello\_device\_t\*)device;
4. **if**(hello\_device) {
5. close(hello\_device->fd);
6. free(hello\_device);
7. }
9. **return** 0;
10. }
12. **static** **int** hello\_set\_val(**struct** hello\_device\_t\* dev, **int** val) {
13. LOGI("Hello Stub: set value %d to device.", val);
15. write(dev->fd, &val, **sizeof**(val));
17. **return** 0;
18. }
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. }
26. read(dev->fd, val, **sizeof**(\*val));
28. LOGI("Hello Stub: get value %d from device", \*val);
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。