

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:

[view plain](#)

```
1. #ifndef ANDROID_HELLO_INTERFACE_H
2. #define ANDROID_HELLO_INTERFACE_H
3. #include <hardware/hardware.h>
4.
5. __BEGIN_DECLS
6.
7. /*定义模块 ID*/
8. #define HELLO_HARDWARE_MODULE_ID "hello"
9.
10. /*硬件模块结构体*/
11. struct hello_module_t {
12.     struct hw_module_t common;
```

```

13. };
14.
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. };
22.
23. __END_DECLS
24.
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：

[view plain](#)

```

1. #define LOG_TAG "HelloStub"
2.
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>
9.
10. #define DEVICE_NAME "/dev/hello"
11. #define MODULE_NAME "Hello"
12. #define MODULE_AUTHOR "shyluo@gmail.com"
13.
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);
17.
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);
21.
22. /*模块方法表*/
23. static struct hw_module_methods_t hello_module_methods = {
24.     open: hello_device_open
25. };
26.
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：

[view plain](#)

```

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));
3.
4.     if(!dev) {
5.         LOGE("Hello Stub: failed to alloc space");

```

```

6.     return -EFAULT;
7. }
8.
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;
15.
16.    if((dev->fd = open(DEVICE_NAME, O_RDWR)) == -1) {
17.        LOGE("Hello Stub: failed to open /dev/hello -- %s.", strerror(errno)
18.        ); free(dev);
19.        return -EFAULT;
20.    }
21.    *device = &(dev->common);
22.    LOGI("Hello Stub: open /dev/hello successfully.");
23.
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:

[view plain](#)

```

1. static int hello_device_close(struct hw_device_t* device) {
2.     struct hello_device_t* hello_device = (struct hello_device_t*)device;
3.
4.     if(hello_device) {
5.         close(hello_device->fd);

```

```

6.     free(hello_device);
7. }
8.
9.     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。