[在Ubuntu上为Android增加硬件抽象层（HAL）模块访问Linux内核驱动程序](http://blog.csdn.net/luoshengyang/article/details/6573809)

标签： [android](http://www.csdn.net/tag/android)[linux内核](http://www.csdn.net/tag/linux%e5%86%85%e6%a0%b8)[module](http://www.csdn.net/tag/module)[struct](http://www.csdn.net/tag/struct)[ubuntu](http://www.csdn.net/tag/ubuntu)

2011-06-28 22:52 96927人阅读 [评论](http://blog.csdn.net/luoshengyang/article/details/6573809#comments)(136) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/luoshengyang/article/details/6573809#report)

http://static.blog.csdn.net/images/category_icon.jpg 分类：

Android（168） http://static.blog.csdn.net/images/arrow_triangle%20_down.jpg

版权声明：本文为博主原创文章，未经博主允许不得转载。

      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)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)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](http://blog.csdn.net/luoshengyang/article/details/6568411)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](http://blog.csdn.net/luoshengyang/article/details/6573809) [copy](http://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](http://blog.csdn.net/luoshengyang/article/details/6573809) [copy](http://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](http://blog.csdn.net/luoshengyang/article/details/6573809) [copy](http://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](http://blog.csdn.net/luoshengyang/article/details/6573809) [copy](http://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。

[在Ubuntu为Android硬件抽象层（HAL）模块编写JNI方法提供Java访问硬件服务接口](http://blog.csdn.net/luoshengyang/article/details/6575988)

标签： [android](http://www.csdn.net/tag/android)[java](http://www.csdn.net/tag/java)[jni](http://www.csdn.net/tag/jni)[server](http://www.csdn.net/tag/server)[ubuntu](http://www.csdn.net/tag/ubuntu)

2011-06-29 22:27 98719人阅读 [评论](http://blog.csdn.net/luoshengyang/article/details/6575988#comments)(107) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/luoshengyang/article/details/6575988#report)

http://static.blog.csdn.net/images/category_icon.jpg 分类：

Android（168） http://static.blog.csdn.net/images/arrow_triangle%20_down.jpg

版权声明：本文为博主原创文章，未经博主允许不得转载。

        在上两篇文章中，我们介绍了如何为Android系统的硬件编写驱动程序，包括如何在Linux内核空间实现内核驱动程序和在用户空间实现硬件抽象层接口。实现这两者的目的是为了向更上一层提供硬件访问接口，即为Android的Application Frameworks层提供硬件服务。我们知道，Android系统的应用程序是用Java语言编写的，而硬件驱动程序是用C语言来实现的，那么，Java接口如何去访问C接口呢？众所周知，Java提供了JNI方法调用，同样，在Android系统中，Java应用程序通过JNI来调用硬件抽象层接口。在这一篇文章中，我们将介绍如何为Android硬件抽象层接口编写JNI方法，以便使得上层的Java应用程序能够使用下层提供的硬件服务。

      一. 参照[在Ubuntu上为Android增加硬件抽象层（HAL）模块访问Linux内核驱动程序](http://blog.csdn.net/luoshengyang/article/details/6573809)一文，准备好硬件抽象层模块，确保Android系统镜像文件system.img已经包含hello.default模块。

      二. 进入到frameworks/base/services/jni目录，新建com\_android\_server\_HelloService.cpp文件：

**USER-NAME@MACHINE-NAME:~/Android$ cd frameworks/base/services/jni**

**USER-NAME@MACHINE-NAME:~/Android/frameworks/base/services/jni$ vi com\_android\_server\_HelloService.cpp**

在com\_android\_server\_HelloService.cpp文件中，实现JNI方法。注意文件的命令方法，com\_android\_server前缀表示的是包名，表示硬件服务HelloService是放在frameworks/base/services/java目录下的com/android/server目录的，即存在一个命令为com.android.server.HelloService的类。这里，我们暂时略去HelloService类的描述，在下一篇文章中，我们将回到HelloService类来。简单地说，HelloService是一个提供Java接口的硬件访问服务类。

      首先是包含相应的头文件：

**[cpp]** [view plain](http://blog.csdn.net/luoshengyang/article/details/6575988) [copy](http://blog.csdn.net/luoshengyang/article/details/6575988)

1. #define LOG\_TAG "HelloService"
2. #include "jni.h"
3. #include "JNIHelp.h"
4. #include "android\_runtime/AndroidRuntime.h"
5. #include <utils/misc.h>
6. #include <utils/Log.h>
7. #include <hardware/hardware.h>
8. #include <hardware/hello.h>
9. #include <stdio.h>

接着定义hello\_init、hello\_getVal和hello\_setVal三个JNI方法：

**[cpp]** [view plain](http://blog.csdn.net/luoshengyang/article/details/6575988) [copy](http://blog.csdn.net/luoshengyang/article/details/6575988)

1. **namespace** android
2. {
3. /\*在硬件抽象层中定义的硬件访问结构体，参考<hardware/hello.h>\*/
4. **struct** hello\_device\_t\* hello\_device = NULL;
5. /\*通过硬件抽象层定义的硬件访问接口设置硬件寄存器val的值\*/
6. **static** **void** hello\_setVal(JNIEnv\* env, jobject clazz, jint value) {
7. **int** val = value;
8. LOGI("Hello JNI: set value %d to device.", val);
9. **if**(!hello\_device) {
10. LOGI("Hello JNI: device is not open.");
11. **return**;
12. }
14. hello\_device->set\_val(hello\_device, val);
15. }
16. /\*通过硬件抽象层定义的硬件访问接口读取硬件寄存器val的值\*/
17. **static** jint hello\_getVal(JNIEnv\* env, jobject clazz) {
18. **int** val = 0;
19. **if**(!hello\_device) {
20. LOGI("Hello JNI: device is not open.");
21. **return** val;
22. }
23. hello\_device->get\_val(hello\_device, &val);
25. LOGI("Hello JNI: get value %d from device.", val);
27. **return** val;
28. }
29. /\*通过硬件抽象层定义的硬件模块打开接口打开硬件设备\*/
30. **static** **inline** **int** hello\_device\_open(**const** hw\_module\_t\* module, **struct** hello\_device\_t\*\* device) {
31. **return** module->methods->open(module, HELLO\_HARDWARE\_MODULE\_ID, (**struct** hw\_device\_t\*\*)device);
32. }
33. /\*通过硬件模块ID来加载指定的硬件抽象层模块并打开硬件\*/
34. **static** jboolean hello\_init(JNIEnv\* env, jclass clazz) {
35. hello\_module\_t\* module;
37. LOGI("Hello JNI: initializing......");
38. **if**(hw\_get\_module(HELLO\_HARDWARE\_MODULE\_ID, (**const** **struct** hw\_module\_t\*\*)&module) == 0) {
39. LOGI("Hello JNI: hello Stub found.");
40. **if**(hello\_device\_open(&(module->common), &hello\_device) == 0) {
41. LOGI("Hello JNI: hello device is open.");
42. **return** 0;
43. }
44. LOGE("Hello JNI: failed to open hello device.");
45. **return** -1;
46. }
47. LOGE("Hello JNI: failed to get hello stub module.");
48. **return** -1;
49. }
50. /\*JNI方法表\*/
51. **static** **const** JNINativeMethod method\_table[] = {
52. {"init\_native", "()Z", (**void**\*)hello\_init},
53. {"setVal\_native", "(I)V", (**void**\*)hello\_setVal},
54. {"getVal\_native", "()I", (**void**\*)hello\_getVal},
55. };
56. /\*注册JNI方法\*/
57. **int** register\_android\_server\_HelloService(JNIEnv \*env) {
58. **return** jniRegisterNativeMethods(env, "com/android/server/HelloService", method\_table, NELEM(method\_table));
59. }
60. };

注意，在hello\_init函数中，通过Android硬件抽象层提供的hw\_get\_module方法来加载模块ID为HELLO\_HARDWARE\_MODULE\_ID的硬件抽象层模块，其中，HELLO\_HARDWARE\_MODULE\_ID是在<hardware/hello.h>中定义的。Android硬件抽象层会根据HELLO\_HARDWARE\_MODULE\_ID的值在Android系统的/system/lib/hw目录中找到相应的模块，然后加载起来，并且返回hw\_module\_t接口给调用者使用。在jniRegisterNativeMethods函数中，第二个参数的值必须对应HelloService所在的包的路径，即com.android.server.HelloService。

      三. 修改同目录下的onload.cpp文件，首先在namespace android增加register\_android\_server\_HelloService函数声明：

      namespace android {

      ..............................................................................................

**int register\_android\_server\_HelloService(JNIEnv \*env);**

      };

      在JNI\_onLoad增加register\_android\_server\_HelloService函数调用：

      extern "C" jint JNI\_onLoad(JavaVM\* vm, void\* reserved)

      {

       .................................................................................................

**register\_android\_server\_HelloService(env);**

.................................................................................................

      }

      这样，在Android系统初始化时，就会自动加载该JNI方法调用表。

      四. 修改同目录下的Android.mk文件，在LOCAL\_SRC\_FILES变量中增加一行：

      LOCAL\_SRC\_FILES:= \

      com\_android\_server\_AlarmManagerService.cpp \

      com\_android\_server\_BatteryService.cpp \

      com\_android\_server\_InputManager.cpp \

      com\_android\_server\_LightsService.cpp \

      com\_android\_server\_PowerManagerService.cpp \

      com\_android\_server\_SystemServer.cpp \

      com\_android\_server\_UsbService.cpp \

      com\_android\_server\_VibratorService.cpp \

      com\_android\_server\_location\_GpsLocationProvider.cpp \

**com\_android\_server\_HelloService.cpp /**

      onload.cpp

      五. 编译和重新找亿system.img：

**USER-NAME@MACHINE-NAME:~/Android$ mmm frameworks/base/services/jni**

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

这样，重新打包的system.img镜像文件就包含我们刚才编写的JNI方法了，也就是我们可以通过Android系统的Application Frameworks层提供的硬件服务HelloService来调用这些JNI方法，进而调用低层的硬件抽象层接口去访问硬件了。前面提到，在这篇文章中，我们暂时忽略了HelloService类的实现，在下一篇文章中，我们将描述如何实现硬件服务HelloService，敬请关注。

[在Ubuntu上为Android系统的Application Frameworks层增加硬件访问服务](http://blog.csdn.net/luoshengyang/article/details/6578352)

标签： [android](http://www.csdn.net/tag/android)[application](http://www.csdn.net/tag/application)[frameworks](http://www.csdn.net/tag/frameworks)[jni](http://www.csdn.net/tag/jni)[ubuntu](http://www.csdn.net/tag/ubuntu)

2011-06-30 23:10 75341人阅读 [评论](http://blog.csdn.net/luoshengyang/article/details/6578352#comments)(89) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/luoshengyang/article/details/6578352#report)

http://static.blog.csdn.net/images/category_icon.jpg 分类：

Android（168） http://static.blog.csdn.net/images/arrow_triangle%20_down.jpg

版权声明：本文为博主原创文章，未经博主允许不得转载。

      在数字科技日新月异的今天，软件和硬件的完美结合，造就了智能移动设备的流行。今天大家对iOS和Android系统的趋之若鹜，一定程度上是由于这两个系统上有着丰富多彩的各种应用软件。因此，软件和硬件的关系，在一定程度上可以说，硬件是为软件服务的。硬件工程师研发出一款硬件设备，自然少了软件工程师为其编写驱动程序；而驱动程序的最终目的，是为了使得最上层的应用程序能够使用这些硬件提供的服务来为用户提供软件功能。对Android系统上的应用软件来说，就是要在系统的Application Frameworks层为其提供硬件服务。在前面的几篇文章中，我们着重介绍了Linux内核层、硬件抽象层和运行时库层提供的自定义硬件服务接口，这些接口都是通过C或者C++语言来实现的。在这一篇文章中，我们将介绍如何在Android系统的Application Frameworks层提供Java接口的硬件服务。

      一. 参照[在Ubuntu为Android硬件抽象层（HAL）模块编写JNI方法提供Java访问硬件服务接口](http://blog.csdn.net/luoshengyang/article/details/6575988)一文所示，为硬件抽象层模块准备好JNI方法调用层。

      二. 在Android系统中，硬件服务一般是运行在一个独立的进程中为各种应用程序提供服务。因此，调用这些硬件服务的应用程序与这些硬件服务之间的通信需要通过代理来进行。为此，我们要先定义好通信接口。进入到frameworks/base/core/java/android/os目录，新增IHelloService.aidl接口定义文件：

**USER-NAME@MACHINE-NAME:~/Android$ cd frameworks/base/core/java/android/os**

**USER-NAME@MACHINE-NAME:~/Android/frameworks/base/core/java/android/os$ vi IHelloService.aidl**

IHelloService.aidl定义了IHelloService接口：

**[java]** [view plain](http://blog.csdn.net/luoshengyang/article/details/6578352) [copy](http://blog.csdn.net/luoshengyang/article/details/6578352)

1. **package** android.os;
3. **interface** IHelloService {
4. **void** setVal(**int** val);
5. **int** getVal();
6. }

IHelloService接口主要提供了设备和获取硬件寄存器val的值的功能，分别通过setVal和getVal两个函数来实现。

三.返回到frameworks/base目录，打开Android.mk文件，修改LOCAL\_SRC\_FILES变量的值，增加IHelloService.aidl源文件：

## READ ME: ########################################################

   ##

   ## When updating this list of aidl files, consider if that aidl is

   ## part of the SDK API. If it is, also add it to the list below that

   ## is preprocessed and distributed with the SDK. This list should

   ## not contain any aidl files for parcelables, but the one below should

   ## if you intend for 3rd parties to be able to send those objects

   ## across process boundaries.

   ##

   ## READ ME: ########################################################

   LOCAL\_SRC\_FILES += /

   ....................................................................

   core/java/android/os/IVibratorService.aidl /

**core/java/android/os/IHelloService.aidl /**

   core/java/android/service/urlrenderer/IUrlRendererService.aidl /

   .....................................................................

    四. 编译IHelloService.aidl接口：

**USER-NAME@MACHINE-NAME:~/Android$ mmm frameworks/base**

   这样，就会根据IHelloService.aidl生成相应的IHelloService.Stub接口。

   五.进入到frameworks/base/services/java/com/android/server目录，新增HelloService.java文件：

**[java]** [view plain](http://blog.csdn.net/luoshengyang/article/details/6578352) [copy](http://blog.csdn.net/luoshengyang/article/details/6578352)

1. **package** com.android.server;
2. **import** android.content.Context;
3. **import** android.os.IHelloService;
4. **import** android.util.Slog;
5. **public** **class** HelloService **extends** IHelloService.Stub {
6. **private** **static** **final** String TAG = "HelloService";
7. HelloService() {
8. init\_native();
9. }
10. **public** **void** setVal(**int** val) {
11. setVal\_native(val);
12. }
13. **public** **int** getVal() {
14. **return** getVal\_native();
15. }
17. **private** **static** **native** **boolean** init\_native();
18. **private** **static** **native** **void** setVal\_native(**int** val);
19. **private** **static** **native** **int** getVal\_native();
20. };

   HelloService主要是通过调用JNI方法init\_native、setVal\_native和getVal\_native(见[在Ubuntu为Android硬件抽象层（HAL）模块编写JNI方法提供Java访问硬件服务接口](http://blog.csdn.net/luoshengyang/article/details/6575988)一文)来提供硬件服务。

     六. 修改同目录的SystemServer.java文件，在ServerThread::run函数中增加加载HelloService的代码：

     @Override

     public void run() {

     ....................................................................................

            try {

                  Slog.i(TAG, "DiskStats Service");

                  ServiceManager.addService("diskstats", new DiskStatsService(context));

            } catch (Throwable e) {

                  Slog.e(TAG, "Failure starting DiskStats Service", e);

            }

**try {**

**Slog.i(TAG, "Hello Service");**

**ServiceManager.addService("hello", new HelloService());**

**} catch (Throwable e) {**

**Slog.e(TAG, "Failure starting Hello Service", e);**

**}**

     ......................................................................................

     }

     七. 编译HelloService和重新打包system.img：

**USER-NAME@MACHINE-NAME:~/Android$ mmm frameworks/base/services/java**

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

这样，重新打包后的system.img系统镜像文件就在Application Frameworks层中包含了我们自定义的硬件服务HelloService了，并且会在系统启动的时候，自动加载HelloService。这时，应用程序就可以通过Java接口来访问Hello硬件服务了。我们将在下一篇文章中描述如何编写一个Java应用程序来调用这个HelloService接口来访问硬件，敬请期待。

[在Ubuntu上为Android系统内置Java应用程序测试Application Frameworks层的硬件服务](http://blog.csdn.net/luoshengyang/article/details/6580267)

标签： [android](http://www.csdn.net/tag/android)[application](http://www.csdn.net/tag/application)[frameworks](http://www.csdn.net/tag/frameworks)[java](http://www.csdn.net/tag/java)[ubuntu](http://www.csdn.net/tag/ubuntu)

2011-07-02 14:35 71016人阅读 [评论](http://blog.csdn.net/luoshengyang/article/details/6580267#comments)(185) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/luoshengyang/article/details/6580267#report)

http://static.blog.csdn.net/images/category_icon.jpg 分类：

Android（168） http://static.blog.csdn.net/images/arrow_triangle%20_down.jpg

版权声明：本文为博主原创文章，未经博主允许不得转载。

        我们在Android系统增加硬件服务的目的是为了让应用层的APP能够通过Java接口来访问硬件服务。那么， APP如何通过Java接口来访问Application Frameworks层提供的硬件服务呢？在这一篇文章中，我们将在Android系统的应用层增加一个内置的应用程序，这个内置的应用程序通过ServiceManager接口获取指定的服务，然后通过这个服务来获得硬件服务。

        一. 参照[在Ubuntu上为Android系统的Application Frameworks层增加硬件访问服务](http://blog.csdn.net/luoshengyang/article/details/6578352)一文，在Application Frameworks层定义好自己的硬件服务HelloService，并提供IHelloService接口提供访问服务。

       二. 为了方便开发，我们可以在IDE环境下使用Android SDK来开发Android应用程序。开发完成后，再把程序源代码移植到Android源代码工程目录中。使用Eclipse的Android插件ADT创建Android工程很方便，这里不述，可以参考网上其它资料。工程名称为Hello，下面主例出主要文件：

    主程序是src/shy/luo/hello/Hello.java：

**[java]** [view plain](http://blog.csdn.net/luoshengyang/article/details/6580267) [copy](http://blog.csdn.net/luoshengyang/article/details/6580267)

1. **package** shy.luo.hello;
3. **import** shy.luo.hello.R;
4. **import** android.app.Activity;
5. **import** android.os.ServiceManager;
6. **import** android.os.Bundle;
7. **import** android.os.IHelloService;
8. **import** android.os.RemoteException;
9. **import** android.util.Log;
10. **import** android.view.View;
11. **import** android.view.View.OnClickListener;
12. **import** android.widget.Button;
13. **import** android.widget.EditText;
15. **public** **class** Hello **extends** Activity **implements** OnClickListener {
16. **private** **final** **static** String LOG\_TAG = "shy.luo.renju.Hello";
18. **private** IHelloService helloService = **null**;
20. **private** EditText valueText = **null**;
21. **private** Button readButton = **null**;
22. **private** Button writeButton = **null**;
23. **private** Button clearButton = **null**;
25. /\*\* Called when the activity is first created. \*/
26. @Override
27. **public** **void** onCreate(Bundle savedInstanceState) {
28. **super**.onCreate(savedInstanceState);
29. setContentView(R.layout.main);
31. helloService = IHelloService.Stub.asInterface(
32. ServiceManager.getService("hello"));
34. valueText = (EditText)findViewById(R.id.edit\_value);
35. readButton = (Button)findViewById(R.id.button\_read);
36. writeButton = (Button)findViewById(R.id.button\_write);
37. clearButton = (Button)findViewById(R.id.button\_clear);
39. readButton.setOnClickListener(**this**);
40. writeButton.setOnClickListener(**this**);
41. clearButton.setOnClickListener(**this**);
43. Log.i(LOG\_TAG, "Hello Activity Created");
44. }
46. @Override
47. **public** **void** onClick(View v) {
48. **if**(v.equals(readButton)) {
49. **try** {
50. **int** val = helloService.getVal();
51. String text = String.valueOf(val);
52. valueText.setText(text);
53. } **catch** (RemoteException e) {
54. Log.e(LOG\_TAG, "Remote Exception while reading value from device.");
55. }
56. }
57. **else** **if**(v.equals(writeButton)) {
58. **try** {
59. String text = valueText.getText().toString();
60. **int** val = Integer.parseInt(text);
61. helloService.setVal(val);
62. } **catch** (RemoteException e) {
63. Log.e(LOG\_TAG, "Remote Exception while writing value to device.");
64. }
65. }
66. **else** **if**(v.equals(clearButton)) {
67. String text = "";
68. valueText.setText(text);
69. }
70. }
71. }

    程序通过ServiceManager.getService("hello")来获得HelloService，接着通过IHelloService.Stub.asInterface函数转换为IHelloService接口。其中，服务名字“hello”是系统启动时加载HelloService时指定的，而IHelloService接口定义在android.os.IHelloService中，具体可以参考[在Ubuntu上为Android系统的Application Frameworks层增加硬件访问服务](http://blog.csdn.net/luoshengyang/article/details/6578352)一文。这个程序提供了简单的读定自定义硬件有寄存器val的值的功能，通过IHelloService.getVal和IHelloService.setVal两个接口实现。

界面布局文件res/layout/main.xml：

**[html]** [view plain](http://blog.csdn.net/luoshengyang/article/details/6580267) [copy](http://blog.csdn.net/luoshengyang/article/details/6580267)

1. **<?xml** version="1.0" encoding="utf-8"**?>**
2. **<LinearLayout** xmlns:android="http://schemas.android.com/apk/res/android"
3. android:orientation="vertical"
4. android:layout\_width="fill\_parent"
5. android:layout\_height="fill\_parent"**>**
6. **<LinearLayout**
7. android:layout\_width="fill\_parent"
8. android:layout\_height="wrap\_content"
9. android:orientation="vertical"
10. android:gravity="center"**>**
11. **<TextView**
12. android:layout\_width="wrap\_content"
13. android:layout\_height="wrap\_content"
14. android:text="@string/value"**>**
15. **</TextView>**
16. **<EditText**
17. android:layout\_width="fill\_parent"
18. android:layout\_height="wrap\_content"
19. android:id="@+id/edit\_value"
20. android:hint="@string/hint"**>**
21. **</EditText>**
22. **</LinearLayout>**
23. **<LinearLayout**
24. android:layout\_width="fill\_parent"
25. android:layout\_height="wrap\_content"
26. android:orientation="horizontal"
27. android:gravity="center"**>**
28. **<Button**
29. android:id="@+id/button\_read"
30. android:layout\_width="wrap\_content"
31. android:layout\_height="wrap\_content"
32. android:text="@string/read"**>**
33. **</Button>**
34. **<Button**
35. android:id="@+id/button\_write"
36. android:layout\_width="wrap\_content"
37. android:layout\_height="wrap\_content"
38. android:text="@string/write"**>**
39. **</Button>**
40. **<Button**
41. android:id="@+id/button\_clear"
42. android:layout\_width="wrap\_content"
43. android:layout\_height="wrap\_content"
44. android:text="@string/clear"**>**
45. **</Button>**
46. **</LinearLayout>**
47. **</LinearLayout>**

字符串文件res/values/strings.xml：

**[html]** [view plain](http://blog.csdn.net/luoshengyang/article/details/6580267) [copy](http://blog.csdn.net/luoshengyang/article/details/6580267)

1. **<?xml** version="1.0" encoding="utf-8"**?>**
2. **<resources>**
3. **<string** name="app\_name"**>**Hello**</string>**
4. **<string** name="value"**>**Value**</string>**
5. **<string** name="hint"**>**Please input a value...**</string>**
6. **<string** name="read"**>**Read**</string>**
7. **<string** name="write"**>**Write**</string>**
8. **<string** name="clear"**>**Clear**</string>**
9. **</resources>**

[http://static.blog.csdn.net/images/save_snippets.png](javascript:;)

程序描述文件AndroidManifest.xml：

**[html]** [view plain](http://blog.csdn.net/luoshengyang/article/details/6580267) [copy](http://blog.csdn.net/luoshengyang/article/details/6580267)

1. **<?xml** version="1.0" encoding="utf-8"**?>**
2. **<manifest** xmlns:android="http://schemas.android.com/apk/res/android"
3. package="shy.luo.hello"
4. android:versionCode="1"
5. android:versionName="1.0"**>**
6. **<application** android:icon="@drawable/icon" android:label="@string/app\_name"**>**
7. **<activity** android:name=".Hello"
8. android:label="@string/app\_name"**>**
9. **<intent-filter>**
10. **<action** android:name="android.intent.action.MAIN" **/>**
11. **<category** android:name="android.intent.category.LAUNCHER" **/>**
12. **</intent-filter>**
13. **</activity>**
14. **</application>**
15. **</manifest>**

[http://static.blog.csdn.net/images/save_snippets.png](javascript:;)

三. 将Hello目录拷贝至packages/experimental目录，新增Android.mk文件：  
    **USER-NAME@MACHINE-NAME:~/Android/packages/experimental$ vi Android.mk**

Android.mk的文件内容如下：

**LOCAL\_PATH:= $(call my-dir)**

**include $(CLEAR\_VARS)**

**LOCAL\_MODULE\_TAGS := optional**

**LOCAL\_SRC\_FILES := $(call all-subdir-java-files)**

**LOCAL\_PACKAGE\_NAME := Hello**

**include $(BUILD\_PACKAGE)**

四. 编译：

**USER-NAME@MACHINE-NAME:~/Android$ mmm packages/experimental/Hello**

编译成功后，便可以在out/target/product/generic/system/app目录下看到Hello.apk文件了。  
    五. 重新打包系统镜像文件system.img：

**USER-NAME@MACHINE-NAME:~/Android$ make snod**  
    重新打包后的system.img文件就内置了Hello.apk文件了。

六. 运行Android模拟器：

**USER-NAME@MACHINE-NAME:~/Android$ emulator -kernel kernel/common/arch/arm/boot/zImage &**

在Home Screen中可以看到Hello应用程序：



打开Hello应用程序：



点击Read按钮，可以从HelloService中读取硬件寄存器val的值；点击Clear按钮，可以清空文本框的值；在文本框中输入一个数值，再点击Write按钮，便可以将这个值写入到硬件寄存器val中去，可以再次点击Read按钮来验证是否正确写入了值。

至此，我们就完整地学习了在Android的Linux内核空间添加硬件驱动程序、在Android的硬件抽象层添加硬件接口、在Android的Application Frameworks层提供硬件服务以及在Android的应用层调用硬件服务的整个过程了，希望能为读者进入Android系统提供入门帮助。重新学习整个过程，请参考[Android硬件抽象层（HAL）概要介绍和学习计划](http://blog.csdn.net/luoshengyang/article/details/6567257)。