在Ubuntu上为Android系统的Application Frameworks层增加硬件访问服务

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]

01. package android.os;
02.
03. interface IHelloService {
04.     void setVal(int val);
05.     int getVal();
06. }
```

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]

01. package com.android.server;
02. import android.content.Context;
03. import android.os.IHelloService;
04. import android.util.Slog;
05. public class HelloService extends IHelloService.Stub {
06.     private static final String TAG = "HelloService";
07.     HelloService() {
08.         init_native();
09.     }
10.     public void setVal(int val) {
11.         setVal_native(val);
12.     }
13.     public int getVal() {
14.         return getVal_native();
15.     }
16.
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。