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：

view plain

```
1.  package android.os;
2.
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：

view plain

```
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.     }
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。