RFID 技术在 Android 系统上的应用实现

赵作人 刘廷龙
(大连工业大学网络信息中心 大连 116034)

摘 要 在物联网的时代背景下,Android 智能手机系统和蓝牙等无线技术的应用范围越来越广。Android 平台下各个功能模块和无线技术已成为当今研究的热点。通过分析 Android 平台下蓝牙自底向上的所有技术细节,并通过蓝牙将 Android 设备与 RFID 设备进行连接使用,来促进 Android 平台下蓝牙技术的发展和应用,进而促进物联网的发展。

关键词 Android 平台,蓝牙技术,BlueZ,RFID 设备
中图法分类号 TP311.52 文献标识码 A

RFID Technology Realization in Android System

ZHAO Zuo-ren LIU Ting-long
(Network Information Center, Dalian Polytechnic University, Dalian 116034, China)

Abstract The applications of Android smart phone system and Bluetooth wireless technology have become more and more widely used in the background of Things to Things. Each module of Android platform and wireless technologies become hot topics of the present study. We analyzed all the bottom-up technical details of Bluetooth of the Android platform, and connected Android device with RFID devices to use by Bluetooth, to promote the development and application of Bluetooth technology in the Android platform, thus to contribute to the development of Things.

Keywords Android platform, Bluetooth technology, BlueZ, RFID equipment

1 引言

Android 的蓝牙模块底层的实现是基于 Linux 操作系统的,包括 Linux 内核和驱动程序。Linux 操作系统位于 Android 软件系统的最底层。

蓝牙是开放式通信规范,而 Linux 是开放源码的操作系统。廉价的设备和免费的软件,促进了蓝牙和 Linux 的发展与融合[1]。BlueZ 是 Linux 的官方蓝牙协议栈,也是目前应用最广泛的协议栈,几乎支持所有已通过认证的蓝牙设备。

RFID 技术利用射频方式进行非接触双向通信,自动识别目标对象并获取相关数据,具有精度高、环境适应能力强、抗干扰强、操作快捷、可识别高速运动的物体,且可同时识别多个标签等优点,因此备受关注[2]。

物联网就是物与物之间的联网。物联网是在互联网基础之上发展而来的,互联网的主体是人,而物联网则将主体扩展到人类生活中具体的物,诸如一辆汽车、一张床、一个微波炉,甚至一间房屋。任何一个物体都可以涵盖在这张网中,从而实现物与物之间、人与物之间的信息交流,达到智能化识别、定位、监控与管理该物体的目的[3]。

2 Android 的蓝牙技术

Android 蓝牙系统的核心围绕 BlueZ 实现。BlueZ 的协议栈在 Linux2.6.X 内核中已经包含。而 BlueZ 的用户空间实现,Android 已经移植并嵌入到自身的平台中,并跟随其更新而更新。

Android 中蓝牙系统的结构如图 1 所示。

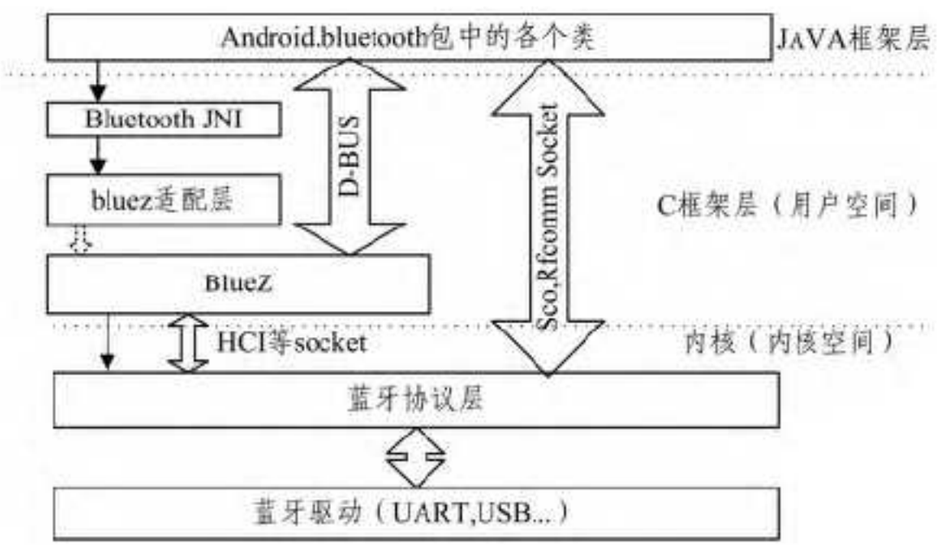

图 1 蓝牙系统组织结构图

从图 1 中可以看出,Android 蓝牙系统涉及 Android 系统自底向上的所有层次

2.1 BlueZ 介绍

BlueZ 目前已成为一个开放性的源码工程。它可以很好地在 Linux 支持的各种体系的硬件平台下运行,包括各种单处理器平台、多处理器平台及超线程系统。BlueZ 由多个独立的模块组成,内核空间主要包括设备驱动层、蓝牙协议及 HCI 层、L2CAP 与 SCO 音频层、RFCOMM、BNEE CMTP 与 HIDP 层、通用蓝牙 SDP 库和后台服务。在用户空间提供了蓝牙配置、测试及协议分析等工具。

赵作人(1980—),男,主要研究方向为计算机网络,E-mail:zhaozr@dlpu.edu.cn;刘廷龙(1988—),男,硕士,主要研究方向为嵌入式系统和射频识别,E-mail:liutinglong45@126.com。

2.2 Linux 内核层

Android 蓝牙模块的内核部分由两个部分组成，分别是蓝牙驱动程序和蓝牙协议栈。

2.2.1 蓝牙驱动

最靠近硬件的一层就是驱动程序。蓝牙设备通常通过 USB、SDIO 或 UART（通用异步接收器/传送器）进行连接。USB、SDIO 通常使用标准的硬件接口和驱动，UART 通常需要使用芯片上的高速串口才能承载蓝牙协议中需要高速传输的部分。BlueZ 同时支持这 3 种传输机制。

蓝牙部分的驱动程序在内核代码的“driver/bluetooth”目录中。

下面来分析具体设备驱动的实现。蓝牙协议中实现的设备驱动很多，比如 hci_vhci.c 为蓝牙虚拟主控制器驱动程序，btuart_es.c 为串口接口主控制器驱动程序，btusb.c 为 USB 接口主控制器驱动程序，btsdio.c 为 SDIO 主控制器驱动程序。这里选择以 USB 设备驱动程序的实现为例进行分析，它的具体实现位于 btusb.c 中。首先，需要一个与 hci_dev 结构体对应的 USB 设备驱动结构体 hci_usb，该结构体通过两个分别指向 hci_dev 结构体和 usb_device 结构体的指针将 HCI 层和 USB 设备层联系起来（usb_device 是 USB 设备描述符，它是所有 USB 设备的根描述符）。USB 蓝牙设备驱动的注册和卸载方法在下面的函数方法中定义：

```
static struct usb_driver btusb_driver;
static int __init btusb_init(void);
static void __exit btusb_exit(void);
module_init(btusb_init);
module_exit(btusb_exit);
```

在上面的函数中，分别通过 usb_register 和 usb_deregister 来注册和卸载 btusb_driver 设备。btusb_driver 中包括了名称为“btusb”以及探测、中断函数。还有一个比较重要的 btusb_table，表示指向驱动所支持的蓝牙设备列表，它是一个全局变量，在 btusb.c 的开始部分定义。

当一个 USB 设备连接到主机时，主机会给这个 USB 设备分配一个 1—127 的设备号，并读取这个设备的描述符，查找适合的驱动程序。如果与 hci_usb 驱动程序注册到 USB 核心的信息匹配，则将设备与 hci_usb 驱动绑定，并立即调用此驱动的设备探测函数（probe）。探测函数的主要任务是：初始化可能用于控制 USB 设备的一些局部结构体，并向内核注册设备信息。

当蓝牙设备被拔出或者断开连接时，需要清理与该设备相关的资源，这就需要通过调用 btusb_disconnect 函数来完成。该函数除了清除相关的接口信息外，还需要在内核中注销设备对应的 HCI 设备并释放其占用的资源。

对 USB 设备的具体操作是：在探测期间向内核注册了 HCI 设备，同时初始化设备操作函数的入口，如下：

```
…
  hdev->open=bfusb_open;
  hdev->close=bfusb_close;
  hdev->flush=bfusb_flush;
  hdev->send=bfusb_send_frame;
  hdev->destruct=bfusb_destruct;
  hdev->ioctl=bfusb_ioctl;
…
```

以上定义了打开 USB、关闭 USB、USB 设备刷新、发送数据帧、设备释放等操作。

UART 通用异步接收/发送装置，是一个并行输入成为串行输出的芯片，通常集成在主板上。在 hci_uart.h 中定义了 Android 平台下蓝牙 HCI UART 驱动。定义了 5 个 IOCTL 用来获得设备信息和向设备发送控制参数：

```
#define HCIUARTSETPROTO _IOW('U',200,int)
#define HCIUARTGETPROTO _IOR('U',201,int)
#define HCIUARTGETDEVICE _IOR('U',202,int)
#define HCIUARTSETFLAGS _IOW('U',203,int)
#define HCIUARTGETFLAGS _IOR('U',204,int)
```

通过它们来实现 I/O 设备的控制。其中 _IOW 表示向驱动程序写入数据命令，_IOR 表示构造从驱动程序中读取数据的命令编号。通过在 hci_uart 结构体中定义的 hci_dev 类型的结构体对象来体现 HCI 对 UART 驱动的控制实现。hci_uart_proto 结构体定义了 UART 协议。UART 协议的注册和卸载通过下面的两种方法来实现：

```
int hci_uart_register_proto(struct hci_uart_proto *p);
int hci_uart_unregister_proto(struct hci_uart_proto *p);
```

通过对上面两种驱动的分析可以看出，蓝牙的驱动程序都将通过标准的 HCI 实现控制。不同硬件的初始化过程可能有一些不同，但是最终都将通过 HCI 来为上层提供一个统一的接口，从而使开发者不受具体硬件设备的影响。

2.2.2 蓝牙协议层

相比驱动程序，Android 的蓝牙协议层位于内核空间的较上层。Linux2.6.X 的内核已经集成了 BlueZ 协议栈。内核中的部分主要包含协议的底层实现，以及负责和硬件的通信，代码在“net/bluetooth”目录中。

在实现蓝牙协议时，一般从两方面来考虑。其一，硬件实现部分，位于 HCI 的下面，即上面提到的底层硬件模块；其二，软件实现部分，位于 HCI 的上面，包括了蓝牙协议栈上层的 L2CAP、RFCOMM、SDP 以及蓝牙的一些应用。

Android 中 BlueZ 蓝牙协议栈架构如图 2 所示[1]。

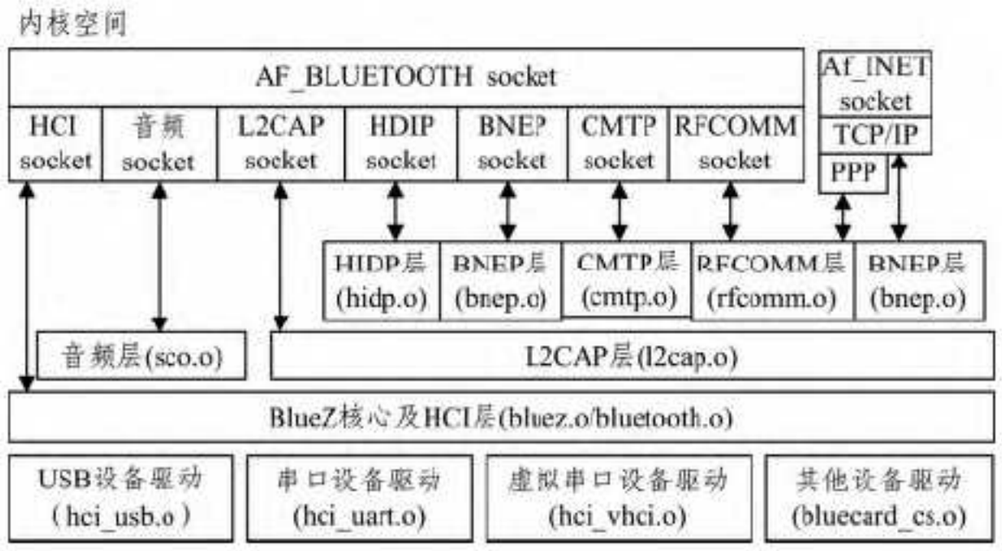

图 2 Android 中 BlueZ 蓝牙协议栈架构

本文只介绍 Android 系统中最重要的两个实现：主机控制接口和串口仿真协议。

(1) 蓝牙主机控制器接口（Host Controller Interface，HCI）由基带控制器、连接管理器、控制和事件寄存器等组成，它是蓝牙协议中软硬件之间的接口，提供了一个调用下层 BB、LMP、状态和控制寄存器的统一命令，上下两个模块接口之间的消息和数据的传递通过 HCI 的解释才能进行。HCI 层以上的协议软件实体运行在主机上，而 HCI 以下的功能由蓝牙设备来完成，二者之间通过传输层进行交换。

HCI 在硬件中的固件与 HCI 在主机端的驱动有多种通信方式，比如 UART、USB 和 PC Card 等。HCI 层在所有的设备面前都被抽象为一个 hci_dev 机构体，因此，无论实际的设备是哪种蓝牙设备以及通过什么方式连接到主机，都需要向 HCI 层和蓝牙核心层注册一个 hci_dev 设备，注册过程由 hci_register_dev() 函数来完成，同时也可以通过 hci_unregister_dev() 函数御载一个蓝牙设备。

hci_dev 结构体的定义位于 hci_core.h 中，主要包含的成员有以下几类：

设备属性，如设备地址、设备名称、设备状态、设备类型和所支持的特性等。

链路属性，主要有支持的分组类型、链路策略和链路模式。

分组属性，如最大传输单元、分组数、已接收分组数等。

任务处理，包含事件的缓存区、等待队列、数据传输的 tasklet 结构体以及相关的锁和信号量。

设备操作，包含 open、close、flush、send、destruct、notify 和 ioctl。

模块所有者，包含私有数据区和异常标志等成员。

由此可见，hci_core.c 相当于一个框架，用于把各种具体的通信方式结合起来，并提供一些公共函数的实现；hci_conn.c 提供了一些连接管理、论证和加密的函数；hci_event.c 为事件处理函数，负责状态机的维护，这些事件通常会使连接从一个状态转换到另一个状态；hci_sock.c 给上层提供了一个 socket 接口，应用程序可以通过 socket 的方式来访问 HCI；hci_sysfs.c 则用来提供一些 sysfs 文件系统接口。另外，l2cap.c 和 sco.c 都是运行在 HCI 之上的协议，分别提供诸如 QoS、分组、多路复用、分段和组装之类的功能以及面向连接的可靠的传输方式，主要用于声音数据传输。具体的上层接口则由 rfcomm、hidp、cmtp 和 bnep 进行封装并实现。

在整个核心框架(hci_core.c)中，通过 hci_cmd_task 来发送 CMD 的任务。它首先从 hdev->cmd_q 队列中取出 CMD，然后调用 hci_send_frame 将 CMD 发送出去，hci_send_frame 又会调用实际的 HCI 驱动的 send 函数发送数据。hci_rx_task 则负责完成接收数据的任务，它从 hdev->rx_q 队列中取出数据，然后根据数据的类型调用上层函数处理。数据包有 3 种类型。

HCI_EVENT_PKT：用于处理一些通信事件，比如建立连接、断开连接、认证和加密等事件，可以通过这些事件来控制协议状态的改变。

HCI_ACLDATA_PKT：异步非连接的数据包，通过 hci_AcLdaTa_packet 提交给上层的 L2CAP 协议处理。

HCI_SCODATA_PKT：同步的面向连接的数据包，通过 hci_scodata_packet 提交给上层的 SCO 协议处理；hci_scodata_packet(hdev, skb)；hci_tx_task 负责完成发送数据的任务，发送所有 connection 中的 ACL 和 SCO 数据以及 hdev->raw_q 中的数据包。另外，HCI 为上层提供了大量的接口来实现相应的功能。

在事件处理过程(hci_event.c)中，hci_si_event 用于发送事件，hci_event_packet 用于处理底层上报的事件。它从 hci_rx_task 处调用。

在 hci_sock.c 中提供具体的操作函数。另外，hci_sock_create 负责创建 sock 函数，并将 sock 的 ops 指向 hci_sock_ops；hci_sock_init/hci_sock_cleanup 用于执行初始化和清理操作。

(2)串口仿真协议(或称线缆替换协议 RFCOMM)是一种仿真有线链路的无线数据仿真协议，并且是符合 ETSI 标准的 TS07.10 串口仿真协议。RFOMM 层支持标准的套接口，并提供了串行仿真 TTY 接口，这使串行端口应用程序和协议可以不加更改地运行在蓝牙设备上。例如，通过点对点协议 PPP 可实现基于 TCP/IP 协议簇的所有网络应用。BNEP 层实现了蓝牙的以太网仿真，TCP/IP 可以直接运行于其上。RFCOMM 协议层位于 L2CAP 协议层与应用层协议之间，为上层提供多路数据链路，指定控制信息和数据信息在通信实体之间交换的规则[4]。

RFCOMM 协议的目的是在 2 个不同设备(通信端点)的应用之间建立并管理一条完整的通信链路，通过命令与响应的交互保持一个可维护的通信会话。RFCOMM 作为 07.10 的子集，在一个串行的异步接口 L2CAP 上允许保持多个同步会话。每个会话可包括不同类型的数据流链路，如语音、数据、传真，各条链路依靠控制信道完成信息参数的协商与流控[5]。RFCOMM 实体的连接模型如图 3 所示。

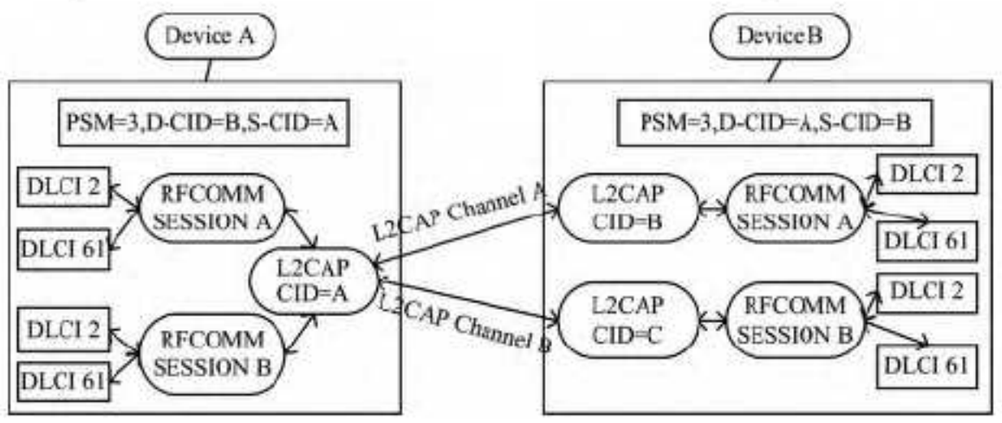

图 3 RFCOMM 实体的连接模型

RFCOMM 帧由 L2CAP 包负责承载，每个包只包含一个 RFCOMM 帧。在图 3 中，D/S-CID 表示 Destination/Source Channel ID；PSM 表示 Protocol Service Multiplexer；DLCI 表示 Data Link Connection Identifier。RFCOMM 连接建立在 L2CAP 逻辑链路的基础上，L2CAP 层向由设备地址 BD_ADDR 表征的两端连接设备提供 CID，以此来表征一条端到端的 L2CAP 逻辑信道。L2CAP 层提供协议复用，其上层协议依据不同的 PSM(Protocol and Service Multiplexor)值做区分。RFCOMM 在 L2CAP 层拥有固定的 PSM=0x0003，使含有 PSM 值的 L2CAP 包可以准确地递交给 RFCOMM 而不是其他复用的协议[5]。因此，PSM 与 CID 共同标识了一条提供给 RFCOMM 的 L2CAP 逻辑连接。RFCOMM 是面向连接的，通过 L2CAP 信道建立一条控制信道，在完成信令的交互握手后，2 个设备间建立起唯一的 RFCOMM 会话。在此会话的基础上，RFCOMM 为上层提供最多 60 个可分优先级的数据链路链接(DLCIx)，使不同的应用层数据通路复用在一个 RFCOMM 会话上，依据 DLCIx 值确定的 server channel 区分不同 RFCOMM 的上层注册服务，提供可靠、有序、不重复的数据收发[6]。

```
static void rfcomm_process_connect(struct rfcomm_session *s);
static inline int rfcomm_process_tx(struct rfcomm_dlc *d);
static int rfcomm_add_listener(bdaddr_t *ba);
```

RFCOMM 协议层命令的交互和数据发送都是通过各种帧完成的。RFCOMM 控制信号和数据封装在不同的帧中进行传输，所使用的帧来自于 GSM TS 07.10 多路控制协议，主要包括：设置异步平衡模式帧(SABM)、无序号响应帧(UA)、断开模式帧(DM)、断开连接帧(DISC)和带头校验的无序号信息帧(UIH)。其中，SABM、DISC 为命令帧；UA、DM 为响应帧。SABM、DISC、UA 和 DM 用于 DLC(包括控制信道 DLCI0)的建立和释放，命令和响应是以锁步次序交换的，在发送命令尚未得到响应之前会阻塞下一帧的发送；UIH 帧分为 2 种：复用控制信道命令在 DICI0 发送，纯数据在特定的 DLCIx 上发送。因此，接收端的处理方式为：通过 Control 字段区分不同的帧类型；通过 Address 域区分不同的 DLCI[7]。

这部分实现在 net/Bluetooth/rfcomm/core.c 和 tty.c 中。其中定义了查找、发现、连接等方法，完成了 rfcomm 协议层的功能。

RFCOMM 通过 bt-sock-register 向上层注册一个 sock。在 net/Bluetooth/rfcomm/sock.c 中定义了一些 rfcomm sock 方法。

2.3 BlueZ 用户空间

2.3.1 BlueZ 库

BlueZ 提供函数库以及应用程序接口，便于程序员开发 Bluetooth 应用程序。BlueZ utils 是主要的工具集，实现对 Bluetooth 设备的初始化和控制。

生成 hcidump 等众多相关工具及库。BlueZ 提供用户空间在蓝牙方面的支持，包括一个主机控制协议(HCI)以及众多内核实现协议的接口，同时，实现蓝牙的所有应用模式(Profile)。在 Android 中，BlueZ 会被编译成如下一些动态链接库和可执行程序。

动态链接库。Libbluetooth：BlueZ 公共库，被各内部应用使用，包含如 SDP、HCI 等协议栈内部接口。Libhcid：主机接口的实现。Audio：音频服务。Input：输入服务。liba2dp：蓝牙立体声服务。

可执行程序。Hcid：直接封装 libhcid，主机接口的实现，同时也是核心守护进程，提供对外的 D-BUS 接口。Hcitool：主机接口辅助工具，主要用 hcitool 来 scan 远端的设备，显示设备地址、名称等。采用表的形式将命令及命令相应函数以及其用途组织起来，通过查找表的形式响应用户输入。用到的函数有 hci-open-dev、hci-ereate-eonnection、hci-read-remote-name 等。Sdptool：提供 SDP 服务相关管理接口，它的服务器 sdpd 被整合到了 hcid 中，主要用来浏览远端设备 SDP 服务，或者管理本地的 SDPD 维护的数据库。它与 hcitool 实现类似，不同的地方在于这些命令使用了一个叫做 search-ontext 的共用结构，不同的命令将其填充，然后调用 sdp 的函数来实现。对于结构通过不同的要求输出格式再重新整理输出。Hciconfig：主机接口辅助工具，用来设置 HCI 设备的参数。

2.3.2 BlueZ 适配层

Android 的 Libray 层还包含了 Android 对 BlueZ 的封装，即 BlueZ 适配层。它实际上就是通过对 BlueZ 和实际硬件驱动进行封装，向上层提供操作 BlueZ 的接口，这样使用 BlueZ 编程将变得更加简单。它生成 bluedroid.so 以及众多相关工具及库[6]，主要实现的功能是对蓝牙设备的管理，它的实现位于"/system/bluetooth"目录中。该部分实现非常简单，主要包括以下内容：

操作硬件的接口实现在"system/Bluetooth/bluedroid/Bluetooth.c"中，主要实现启动蓝牙、禁用蓝牙、检查是否启用。这些实现都很简单，这里不再给出代码。实现过程中还需要几个辅助函数，包括 set-bluetooth-power、check-bluetooth-power、create-hci-sock 等。一些配置文件放在"system/Bluetooth/data/"目录中，将会被复制到"/etc/Bluetooth/"中，比如：Audio.conf 是对音频服务的配置，input.conf 是对输入服务的配置。除此之外，"system/bluetooth"中还包含部分工具实现和测试代码，测试代码可以帮助我们学习如何操作 BlueZ 等功能。

2.4 蓝牙的 JNI 和 java 框架部分

Android-bluetooth-* 以及 android-server-bluetooth* 命名的源代码文件和头文件提供 android.bluetooth 包中几个类和 android.server 包中与 Bluetooth 相关的几个类的支持，生成的内容是 Android 的 JNI 库 libandroid-runtime.so 的一部分[8]。蓝牙部分的 JNI 源代码文件如下所示：

frameworks/base/core/jni/android-bluetooth-* -cpp

frameworks/base/core/jni/android-server-bluetooth*

它们将提供 android.bluetooth 包中多个类的支持，这些内容会与其他模块 JNI 部分一起生成 libandroid-runtime.so。BluetoothAdapter 实现蓝牙功能的开启/关闭、设备的发现和配对、服务的发现和绑定等功能。

Android-server-BluetoothService.cpp 对其提供 JNI 本地代码支持。开关功能通过直接调用 bluez 适配层提供的开关功能来实现。服务发现和绑定等功能，均通过 D-BUS 调用 bluez 接口来实现。BluetoothAdapter 中其他的访问服务流程与此类似。

Android 蓝牙 java 部分和底层的关系如图 4 所示。

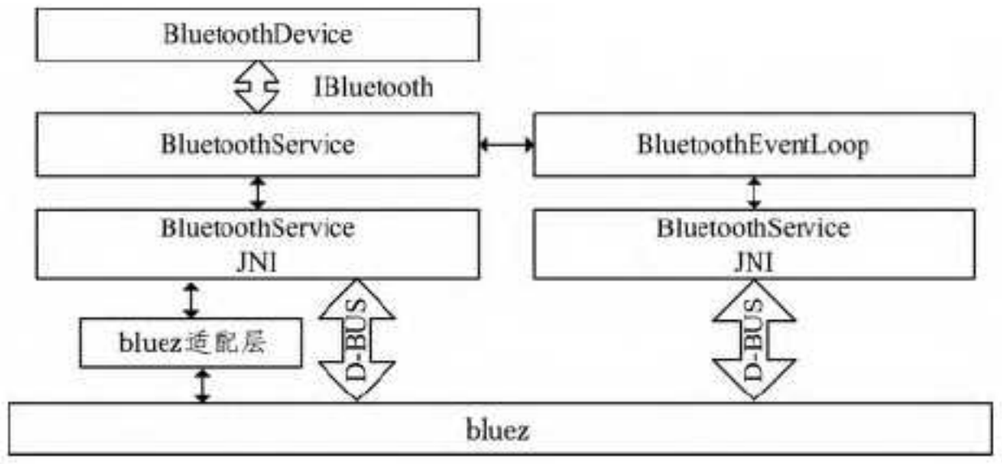

图 4 Android 部分 java 与底层的关系

下面以发现服务的流程为例，展示 BluetoothDevice 与 D-BUS 通信的流程。

客户端 BluetoothDevice.getRemoteServiceChannel 调用服务器端 BluetoothDeviceService.getRemoteServiceChannel，然后到 JNI 的 getRemote-ServiceChannelNative。getRemoteServiceChannelNatice 使用下面代码访问 D-BUS：

ret=dbus-func-args-async(env,nat->conn,-1,onCreateDeviceResult,context-address,eventLoopNat,get-adapter-path(env,object),DBUS-ADAPTER-IFACE,"CreateDevice",DBUS-TYPE-STRING,&c-address,DBUS-TYPE-INVALID);

dbus-func-args-async 实现在 android-bluetooth-com-

mon. cpp 中最终会调用 dbus-func-args-async-valist 运行 D-BUS 调用。

android-bluetooth-eommon. cpp 定义了一个基本的 D-BUS 远程调用流程。

在处理过程中，首先通过 dbus-message-new-method-call 创建一个消息结构 msg。在函数参数中，path 为 mat->adapter，指向 bluez 中注册的蓝牙设备，android-bluetooth-common. h 中使用了硬编码方式，写成了“/hci0”，也就是本机的首个蓝牙设备。如下：

define BLUEZ-ADAPTER-OBJECT-NAME BLUEZ-DBUS-BASE-PATH “hci0”

Ifc 为 DBUS-CLASS-NAME，指向“org. bluez. Adapter”直接访问 bluez 的 HCI 部分的 Adapter 模块接口。Func 为“GetRemoteServiceChannel”，也就是远程调用的函数名。然后通过 dbus-connectin-send-with-reply 即可发送消息，并获取一个 DBusPendingCall 指针 * call。Conn 技术 nat->conn，在 BluetoothDeviceServiice 初始化时通过如下方式获取：

Nat->conn = dbus-bus-get(DBUS-BUS-SYSTEM, &err);

Reply 仅仅表示成功与否的 boolean，而 Call 指向的结构才是用来获取返回信息的关键结构。通过 dbus-pending-eall-set-notify 注册回调函数，当远程函数返回或者超过时会调用此回调函数。

通过 dbus-func-args-async-callback 发现，最终的回调是调用到 pending 里面注册的 pending->user-eb，dbus-func-args-async-callback 的参数即 pending 指针。发送和接收消息后，都需要 dbus-message-unref(msg)。

此处注册的回调函数是 onGetRemoteServiceChannelResult，其在 D-BUS 函数调用完成后被调用。它的实现是调用 BluetoothDevice. getRemoteServiceChannel 执行时注册的回调函数，最终回调到客户端的处理函数，以获取远程服务。与 BluetoothDevice 支持的远程调用再回调的方式相反，BluetoothEventLoop 则是框架从 bluez 获取消息的另一个主要途径。它的核心函数是一个获取 D-BUS 消息的循环，通过 BluetoothEventLoop. start 其他的新线程 Thread(“Bluetooth Event Loop”). run 实现。BluetoothEventLoop. start 则由 BluetoothService. endable 调用，即蓝牙功能开启时开启此循环。循环通过设置消息过滤器来处理每一条来自 D-BUS 的消息。setUpEventLoop()是一个有 JNI 提供的方法，它进一步通过以下的调用来实现。

if(!dbus-eonnection-add-filter(nat->conn, event-filter, nat, NULL)){return JNI-FALSE;}

注册消息的处理函数 event-filter，并使用 dbus-bus-add-match 配置 filter 来匹配每一条感兴趣的消息。Event-filter 完成实际的消息处理并调用响应的处理函数，在 JNI 中实现，通过 JNI 函数指针访问在 java 中实现的回调函数。

3 Android 设备和 RFID 设备的连接

本应用中实现了两个类：ComRfid. java 和 BluetoothConRfid. java。其中 ComRfid. java 对于不同的 RFID 设备有所不同，它主要完成对 RFID 设备的控制操作。本应用使用的 RFID 读写器是 RS-9 Rev. C，全面支持对 HF 频段 ISO15693/ISO18000-3、ISO14443A/B/C 等多种标准协议 RFID 电子标签的读写操作；提供无线蓝牙通讯接口，可与掌上电脑、PDA、智能手机等任何内置蓝牙通讯的手持设备配合使用，不受设备有线接口形式的限制。标签使用的是 S50 非接触式 IC。

BluetoothConRfid. java 完成 Android 的界面显示，蓝牙设备的查找、连接、数据的读写、命令的控制、蓝牙设备的释放等操作。系统运行效果如图 5 所示。

图 5 Android 设备和 RFID 连接应用

OFF/On 是用来关闭/开启蓝牙的开关；搜索设备按钮是用来搜索蓝牙设备的。点击后，在上部的 item1 栏位会显示目前的蓝牙设备，设备名为 BOLUTEK，状态为“未配对”或是“已配对”；点击 Item1 栏位，等待大约 2s 后，会显示“已连接 BOLUTEK 设备”。此时，就可以操作下面对于 rfid 卡操控的按钮，读写的结果会在下面的 Item1 栏位显示。

结束语 随着 Android 系统的发展，其应用越来越广，对蓝牙技术要求也越来越高。通过深入分析 Android 平台下蓝牙技术的各个层次内容，为以后在其平台上该功能的研究和应用积累了经验，也为将来在物联网上实现更为完善的蓝牙功能打下了较好的基础。

参 考 文 献

[1] 梁军学，都滨. Linux 蓝牙协议栈的 USB 设备驱动[J]. 计算机工程，2008，34(9)：273-275

[2] 李军怀，张果谋，于蕾，等. 基于虚拟信号强度的 RFID 定位方法研究[J]. 计算机科学，2012，39(4)：67-70

[3] 邓方源，景小平. 基于物联网的低成本食品跟踪技术的应用研究[J]. 计算机科学，2011，38(10A)：26-29

[4] Bluetooth Special Interest Group. RFCOMM with TS07. 10[Z]. 2003

[5] 刘凯，高强，王尧. 蓝牙 RFCOMM 协议层的研究与实现[J]. 计算机工程，2011，37(12)：92-94

[6] 余胜生，苏汉华，周敬利. Bluetooth 协议栈 RFCOMM 协议层分析与设计[J]. 小型微型计算机系统，2002，23(9)：1037-1040

[7] 马晋兴，陈启军. 蓝牙协议栈中 RFCOMM 协议层的分析与实现[J]. 计算机工程，2004，30(8)：112-113，125

[8] 韩超，梁泉. Android 系统原理及开发要点详解[M]. 北京：电子工业出版社，2010：281-286