**Android蓝牙架构系统**

蓝牙(Bluetooth)提供移动设备及其配件之间低功耗、低成本的无线通信连接。蓝牙技术的核心，是根据蓝牙标准所实现的蓝牙协议栈。蓝牙能同时进行异步数据及同步语音传输，它的底层协议层包括众多协议，如逻辑链路控制和适配协议(L2CAP)、服务发现协议(SDP)、串口模拟协议(RFCOMM)等，一起为上层提供传输支持。而主要控制接口则由主机控制接口层(HCI)体现，它是蓝牙协议中软硬件之间的接口。在各种协议及HCI之上，是蓝牙的上层应用框架，框架里每个应用模式称之为一个Profile，供给外部不同的设备访问，如无线立体声耳机使用的A2DP(Advanced Audio Distribution Profile)以及打印机使用的无线打印机服务等，服务有数十种之多。

1.蓝牙基本架构

Android提供了对蓝牙的支持。蓝牙系统的内容包括：

·Linux内核的蓝牙驱动程序

·Linux内核的蓝牙协议层

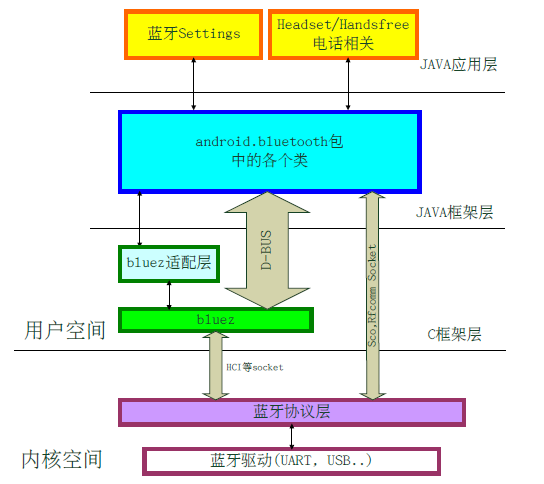
·bluez(蓝牙在用户空间的库)

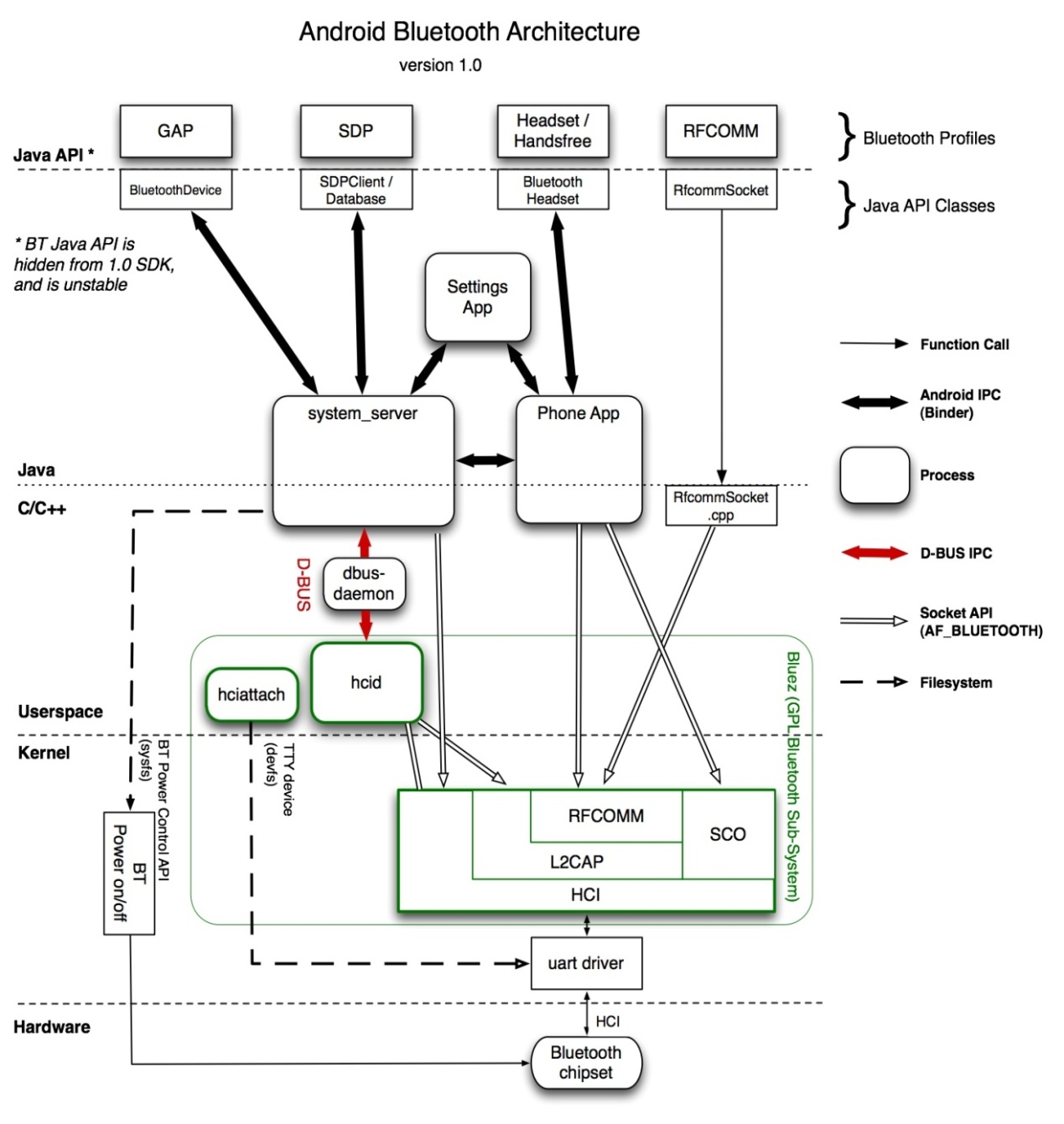
·bluez适配层

·android.bluetooth包中的各个类(蓝牙在框架层的内容)

·蓝牙相关的应用程序

Android的蓝牙部分的结构如下：

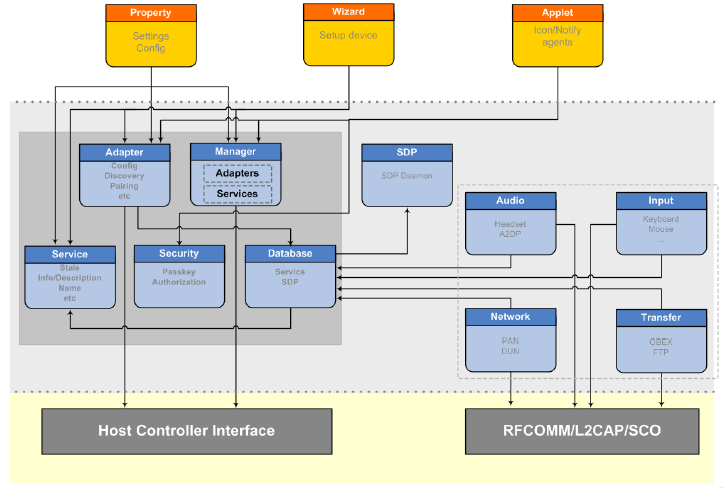
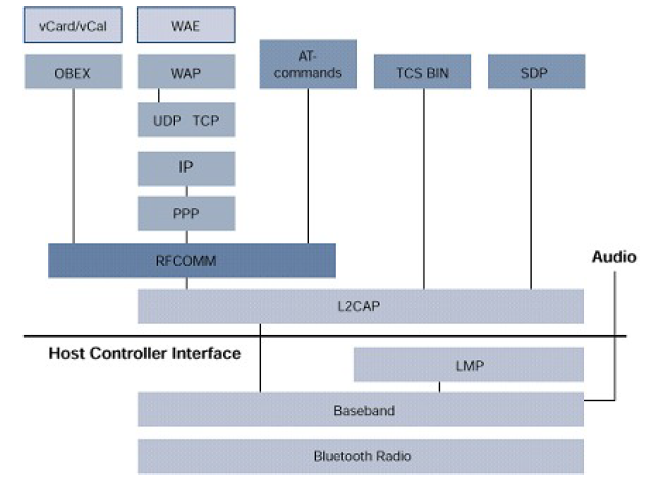


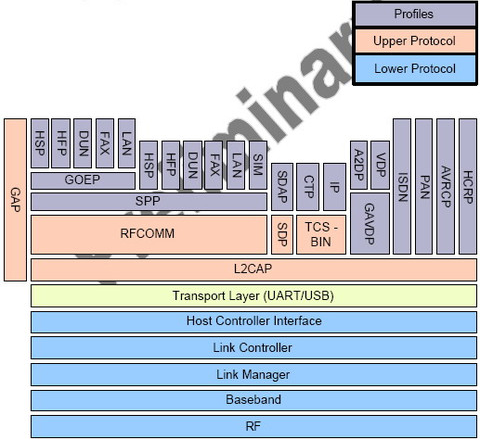


2.蓝牙用户空间库bluez

Android所采用的蓝牙用户空间库是bluez，它是Linux平台上一套完整的蓝牙协议栈开源实现。在Android中，bluez提供了很多分散的应用，包括守护进程和一些工具；bluez通过D-BUS IPC机制来提供应用层接口。需要注意的是，bluez的底层协议实现在kernel代码中，并不属于用户空间。

D-BUS是一套应用广泛的进程间通信机制，相对于Socket等底层IPC，它是更加复杂的IPC机制，支持更系统化的服务名、函数名等，同时也能对众多服务进程和客户端进行管理，调度通信消息的传递。D-BUS和Binder类似，但是应用范围更广泛。

bluez的主要体现结构如下： 



3.bluez适配层

bluez在Android中的使用，需要经过Android的bluez适配层的封装来实现，bluez适配层源代码及头文件路径为：system/bluetooth/。该目录除了包含生成适配层库libbluedroid.so的源码之外，还包含bluez头文件、bluez配置文件等目录。由于bluez使用D-BUS作为与上层沟通的接口，因此适配层构造比较简单，封装了蓝牙的开关功能以及射频开关。

4.蓝牙的JNI和Java部分

蓝牙的JNI源代码文件为：frameworks/base/core/jni/android\_bluetooth\_\*.cpp。蓝牙部分的JNI带有android\_bluetooth\_前缀，它们将提供android.bluetooth包中多个类的支持，这些内容会与其它模块JNI部分一起生成libandroid\_runtime.so。

蓝牙的Java源代码路径为：

·frameworks/base/core/java/android/server：蓝牙的服务部分

·frameworks/base/core/java/android/bluetooth/：Android的蓝牙类

Android支持蓝牙设备的发现、配对、服务查找等基础功能，服务方面支持耳机(Headset)、免提(Handsfree)和立体声(A2DP)等部分。

①核心组成

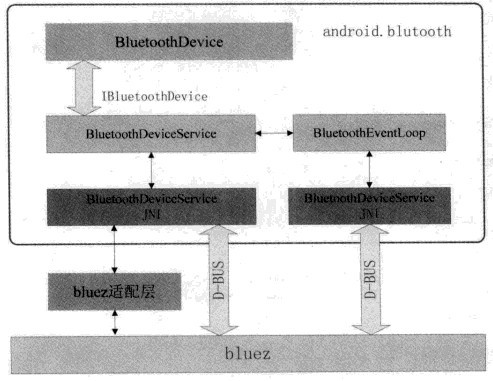
框架核心组成部分是android.bluetooth包中以IBluetoothDevice.aidl为接口的IBinder服务器端BluetoothDeviceService、客户端BluetoothDevice以及android.server包中的BluetoothEventLoop。

BluetoothDeviceService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\_BluetoothDeviceService.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

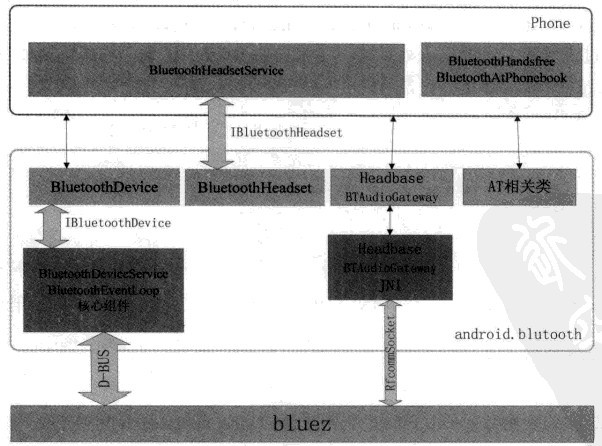
·服务发现和绑定等功能，均通过D-BUS调用bluez接口来实现。BluetoothDevice中其它的访问服务流程与此类似。

Android蓝牙Java部分和底层的关系如下：



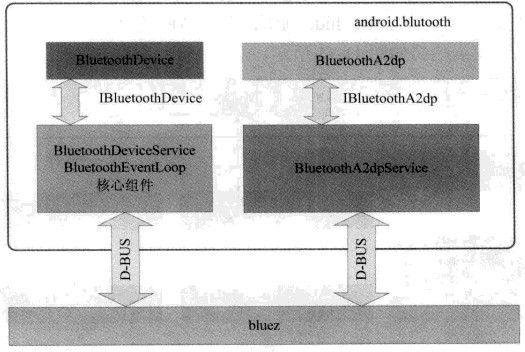
②耳机及免提服务

对于服务实现部分的代码，主要是耳机和免提。两者都支持通话，区别在于免提可以支持一些操作按键，如拒接电话等，而耳机则不行。Android要支持这两个服务，则需要支持两种途径：一是通过SDP发布这两个服务，让配对的蓝牙设备可以发现这两个服务并主动进行连接；二是Android通过SDP判断设备是否支持连接到这两个服务，如果支持则主动去连接设备。连接方式都是通过RFCOMM协议完成的。最终的音频通道数据则由SCO协议传输。Android蓝牙的耳机及免提服务的结构如下：



③A2DP服务

A2DP服务对蓝牙立体声耳机提供支持，相比SCO，它能传输更好音质的音频数据。A2DP更多地用在音乐播放方面，而SCO则更多地用在电话语音等方面。A2DP结构如下：



bluez对A2DP的支持很好，A2DP部分主要围绕IBluetoothA2DP来实现，Service部分通过D-BUS与bluez的Audio部分通信，主要的控制接口是Audio的sink组件，sink用来连接支持A2DP的Headset设备。如果Headset设备支持A2DP，Android可以自动完成SCO和A2DP的切换。

**Android蓝牙系统的源码目录结构：**

1. App部分主要包括OPP和PBAP以及HFP，当然还有settings部分，代码分布如下：

(1) /packages/app/Bluetooth/src/com/android/bluetooth/opp

(2) /packages/app/Bluetooth/src/com/android/bluetooth/pbap

(3) /packages/app/Phone/src/com/android/phone/ 中和BT相关的部分，主要是HFHS打电话相关的

(4) /packages/app/Settings/src/com/android/settings/bluetooth/ 主要是蓝牙开启，搜索，配对等

2. Framework部分主要包括BluetoothAdapter,BluetoothService，BluetoothEventLoop，BluetoothA2dpService等核心类，代码分布如下：

(1)/frameworks/base/core/java/android/server/ 包含BluetoothService，BluetoothEventLoop等核心类，但是并没有向用户应用程序提供接口，属于系统的类。

(2)/frameworks/base/core/java/android/bluetooth 包含了BluetoothAdapter, BluetoothSocket, BluetoothServerSocket等一系列类，这些类是系统向应用程序提供的接口，编写蓝牙相关的应用程序时会用到这些类，检查BT是否打开，搜索设备等等。如果编写蓝牙应用程序，需要熟悉该部分类的功能。

(3) /frameworks/base/core/jni/ 中包含Bluetooth相关的文件，如 android\_server\_BluetoothService.cpp 该部分主要是java层和c/c++层进行通信的

3. Bluez部分，主要包括各个协议的实现，如A2dp，AVRCP，AVDTP，HID，HDP，PAN，DUN以及对BT host的实现，通过该部分向BT Controler部分发送HCI命令，代码分布服下：

/external/bluetooth/bluez (bluez协议层)

4. BluetDroid部分，主要用于开关蓝牙，芯片上电部分(不属于驱动，可以算是硬件抽象层(HAL))， 代码分布如下：  
/system/bluetooth/ (bluez适配层)

5. kernel部分，主要包括RFCOMM，L2CAP等协议以及HCI的实现，代码分布如下：

/kernel/net/bluetooth/(bt底层协议层以及HCI设备抽象层)

6. driver部分，该部分代码还真没了解过，代码分布：

/kernel/drivers/bluetooth/(bt驱动)

另外，driver部分和具体芯片密切相关，有时在下面的目录下也会有：

/vendor/

/device/

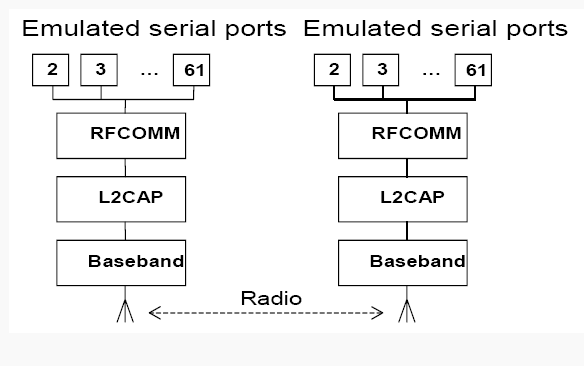
**Android 蓝牙术语及相关概念**

RFCOMM

一个基于[欧洲](http://baike.baidu.com/view/3622.htm)电信标准协会ETSI07.10规程的串行线性仿真协议。此协议提供RS232控制和状态信号，如基带上的损坏，CTS以及[数据](http://baike.baidu.com/view/38752.htm)信号等，为上层业务（如传统的串行线缆应用）提供了传送能力。

RFCOMM是一个简单[传输协议](http://baike.baidu.com/view/441895.htm)，其目的为了解决如何在两个不同设备上的应用程序之间保证一条完整的通信路径，并在它们之间保持一通信段的问题。

RFCOMM协议提供对基于L2CAP协议的串口仿真，基于ETSI07.10。可支持在两个BT设备之间同时保持高达60路的通信连接。



SPP Serial Port Profile

SPP是一个[传输协议](http://baike.baidu.com/view/441895.htm)子集，它规定了在两个对等设备之间建立RFCOMM通信所需要的基本操作。串口应用基于通用访问应用GAP和服务发现应用[SDAP](http://baike.baidu.com/view/493703.htm)应用模型之上，又是其他许多应用框架的基础。SPP可以直接映射到RFCOMM协议中，当两个设备间需要通过RFCOMM仿真的串口进行通信时，SPP对其中必备的服务和过程做出详细的规定。

UUID

通用唯一识别码 (Universally Unique Identifier), 一个[软件](http://baike.baidu.com/view/37.htm)建构的标准，也是被[开源软件](http://baike.baidu.com/view/444964.htm)基金会 (Open Software Foundation, OSF) 的组织应用在[分布式计算环境](http://baike.baidu.com/view/185350.htm) (Distributed Computing Environment, DCE) 领域的一部分。

标准的UUID格式为：xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxx (8-4-4-4-12)。

UUID由以下几部分的组合：

（1）当前日期和时间，UUID的第一个部分与时间有关，如果你在生成一个UUID之后，过几秒又生成一个UUID，则第一个部分不同，其余相同。

（2）时钟序列。

（3）全局唯一的IEEE机器识别号，如果有网卡，从网卡MAC地址获得，没有网卡以其他方式获得。

SDP

服务发现协议(Service Discovery Protocol)是蓝牙技术框架中非常重要的一个部分，它是所有用户模式的基础。使用SDP，可以查询到设备信息和服务类型，之后，蓝牙设备之间的连接才能建立。因此，你不能奢望通过蓝牙耳机打电话，因为它不能提供这种服务。

[A2DP](A2DP介绍.ppt)

[蓝牙](http://baike.baidu.com/view/1028.htm)音频传输模型协定(Advanced Audio Distribution Profile)，定义了在ACL(Asynchronous Connectionless 异步无连接)信道上进行高质量的音频分发的协议和过程，最大的特点是可以直接通过蓝牙立体声耳机听手机中所播放的音乐。

比特率 = 采样率\*采样位数\*通道数;

8000\*7 = 56kbps 通话音质

L2CAP

[逻辑](http://baike.baidu.com/view/1838.htm)链路控制和适配[协议](http://baike.baidu.com/view/36190.htm)（Logical Link Control and Adaptation Protocol），是蓝牙系统中的核心协议，负责适配基带中的上层协议。它同链路管理器并行工作，向上层[协议](http://baike.baidu.com/view/36190.htm)提供定向连接的和无连接的数据业务。这个上层具有L2CAP的分割和重组功能，使更高层次的[协议](http://baike.baidu.com/view/36190.htm)和应用能够以64KB的长度发送和接收[数据包](http://baike.baidu.com/view/25880.htm)。它还能够处理[协议](http://baike.baidu.com/view/36190.htm)的多路复用，以提供多种连接和多个连接类型（通过一个空中接口），同时提供服务质量支持和成组通讯。

[逻辑链路控制](http://baike.baidu.com/view/195067.htm)和适配[协议](http://baike.baidu.com/view/36190.htm)（L2CAP）是基带的上层协议，可以认为它与[LMP](http://baike.baidu.com/view/1217054.htm)并行工作，它们的区别在于当业务数据不经过LMP时，L2CAP为上层提供服务。L2CAP向上层提供面向连接的和无连接的数据服务，它采用了多路技术、分割和重组技术、群提取技术。L2CAP允许高层[协议](http://baike.baidu.com/view/36190.htm)以64K字节收发数据分组。虽然基带[协议](http://baike.baidu.com/view/36190.htm)提供了SCO和ACL两种连接类型，但L2CAP只支持ACL.

LMP

Link Manager Protocol链路管理层，连接管理协议负责蓝牙各设备间连接的建立。

首先，它通过连接的发起、交换、核实，以进行身份认证和加密等安全措施；其次它通过设备间协商以确定基带数据分组的大小；另外，它还可以控制无线部分的电源模式和工作周期，以及微微网内各设备的连接状态。

Android蓝牙功能编译相关

Android蓝牙功能相关:

./device/sprd/sp8825ea/BoardConfig.mk

BOARD\_HAVE\_BLUETOOTH :=true

./device/sprd/sp8825gb/sprd\_core.mk

ifeq ($(BOARD\_SUPPORT\_FEATURE\_BT), true)

PRODUCT\_PACKAGES += Bluetooth

endif

./device/sprd/sp8825ea/init.rc

./device/sprd/sp8825ea/init.sc8825.rc

./device/sprd/sp8825ea/scripts/ext\_chown.sh

./kernel/arch/arm/configs/sp8825ea-vlx\_defconfig

蓝牙驱动现相关：

驱动

./kernel/drivers/bluetooth/Makefile

协议

./kernel/net/bluetooth/cmtp/Makefile

./kernel/net/bluetooth/hidp/Makefile

./kernel/net/bluetooth/Makefile

./kernel/net/bluetooth/bnep/Makefile

./kernel/net/bluetooth/rfcomm/Makefile

博通4330相关文件

driver

  ./ kernel/drivers/net/wireless/bcmdhd

firmware:

./device/sprd/sp8825ea/base.mk

$(call inherit-product, device/sprd/partner/brcm/device-bcm-wlan-b2-sdio.mk)

./hardware/broadcom/wlan/bcmdhd/firmware/bcm4330/\*b2.bin

nv :  ./hardware/broadcom/wlan/bcmdhd/firmware/bcm4330/bcmdhd\_26MHz\_sdio.ca

手机主动连接蓝牙耳机过程

----BluetoothBondState: setBondState (BluetoothDevice.BOND\_BONDED)

----BluetoothService: addProfileState new BluetoothDeviceProfileState

----BluetoothDeviceProfileState: getProfileProxy

----BluetoothAdapter: getProfileProxy new BluetoothHeadset

----BluetoothDeviceProfileState: BondedDevice. enter (InitialState)

log: BluetoothDeviceProfileState: Entering ACL Connected state with: -2 (SM\_INIT\_CMD)

----BluetoothDeviceProfileState: BondedDevice. processMessage

log: BluetoothDeviceProfileState: Device:00:1F:82:39:85:EB Message:ACL Connected State -> Processing Message: -2

----BluetoothService: setProfilePriorities

----BluetoothHeadset: setPriority

----BluetoothBondState: sendBroadcast (BluetoothDevice.ACTION\_BOND\_STATE\_CHANGED)

----BluetoothEventManager: BondStateChangedHandler onBondingStateChanged

----CachedBluetoothDevice: onBondingStateChanged (BluetoothDevice.BOND\_BONDED)

----CachedBluetoothDevice: connect

----CachedBluetoothDevice: connectWithoutResettingTimer

log: BluetoothHeadset: getPriority(00:1F:82:39:85:EB)

----CachedBluetoothDevice: connectInt

----headsetprofile: connect

log: BluetoothHeadset: getConnectedDevices()

----BluetoothHeadset: connect

----BluetoothHeadsetService: mBinder. connect

----BluetoothService: connectHeadset (BluetoothDeviceProfileState.CONNECT\_HFP\_OUTGOING)

----BluetoothProfileState: sendMessage

----BluetoothDeviceProfileState BondedDevice. enter

log: BluetoothDeviceProfileState: Device:00:1F:82:39:85:EB Message:ACL Connected State -> Processing Message: 1(CONNECT\_HFP\_OUTGOING)

----BluetoothDeviceProfileState: BondedDevice.processMessage

log: Device:00:1F:82:39:85:EB Message:ACL Connected State -> Processing Message: 1

----BluetoothDeviceProfileState: Entering OutgoingHandsfree state with: 1 (CONNECT\_HFP\_OUTGOING)

----BluetoothHeadset: connectHeadsetInternal

----BluetoothHeadsetService: connectHeadsetInternal

----BluetoothHeadsetService: getSdpRecordsAndConnect

log: Bluetooth HSHFP: SDP UUID: TYPE\_HANDSFREE

----BluetoothHeadsetService: RfcommConnectThread. waitForConnect

----android\_bluetooth\_HeadsetBase: waitForAsyncConnectNative

----android\_bluetooth\_HeadsetBase: connectAsyncNative

log: BT HSHFP: Created RFCOMM socket fd 142.

----BluetoothHeadsetService: mConnectingStatusHandler. handleMessage RFCOMM\_CONNECTED

----BluetoothHandsfree:connectHeadset

----BluetoothHandsfree:initializeHandsfreeAtParser

if (mHeadsetType == TYPE\_HEADSET) {

initializeHeadsetAtParser();

} else {

initializeHandsfreeAtParser();

}

----BluetoothHeadsetService: audioOn(isIncallAudio)

----BluetoothHeadsetService: connectScoThread (createScoSocket)

----BluetoothHeadsetService: setBluetoothScoOn

蓝牙耳机主动连接手机

----BluetoothHeadsetService: onStart

----BluetoothAudioGateway: start

----android\_bluetooth\_BluetoothAudioGateway: waitForHandsfreeConnectNative

log: BluetoothAudioGateway.cpp: Accepting HF connection.

----android\_bluetooth\_BluetoothAudioGateway: do\_accept

log: BluetoothAudioGateway.cpp: Successful accept() on AG socket 151: new socket 156, address 50:C9:71:42:42:F8, RFCOMM channel 10

----BluetoothAudioGateway: new IncomingConnectionInfo (MSG\_INCOMING\_HANDSFREE\_CONNECTION)

----BluetoothHeadsetService: mIncomingConnectionHandler

----BluetoothService: notifyIncomingConnection (BluetoothDeviceProfileState.CONNECT\_HFP\_INCOMING)

----BluetoothDeviceProfileState: IncomingHandsfree

----BluetoothDeviceProfileState: processIncomingConnectCommand

----BluetoothHeadsetService: createIncomingConnect

----BluetoothHeadsetService: mConnectingStatusHandler. handleMessage (RFCOMM\_CONNECTED)

A2DP

----BluetoothService: sendConnectionStateChange

----BluetoothA2dpService: connect

----BluetoothService:connectSink

----AudioFlinger: setParameters

----AudioPolicyService: setDeviceConnectionState

----AudioPolicyManagerBase: handleA2dpConnection