Android 的连接部分

Android 的连接部分

- 第一部分　WIFI 部分
- 第二部分　蓝牙部分
- 第三部分　GPS 和定位部分

第一部分 WIFI 部分

Wifi （Wireless Fidelity），是使用了 IEEE 的 802.11 协议的无线局域网（Wlan）技术。

在 android 中 Wifi 包括 kernel 的支持和用户空间的程序和库两个部分。

第一部分 WIFI 部分

1.1 WIFI 的基本架构

1.2 WIFI 的本地实现

1.3 WIFI 的 JNI 和 JAVA 层次

1.4 Setting 中的 WIFI 设置

1.5 WIFI 的流程

1.1 WIFI 的基本架构

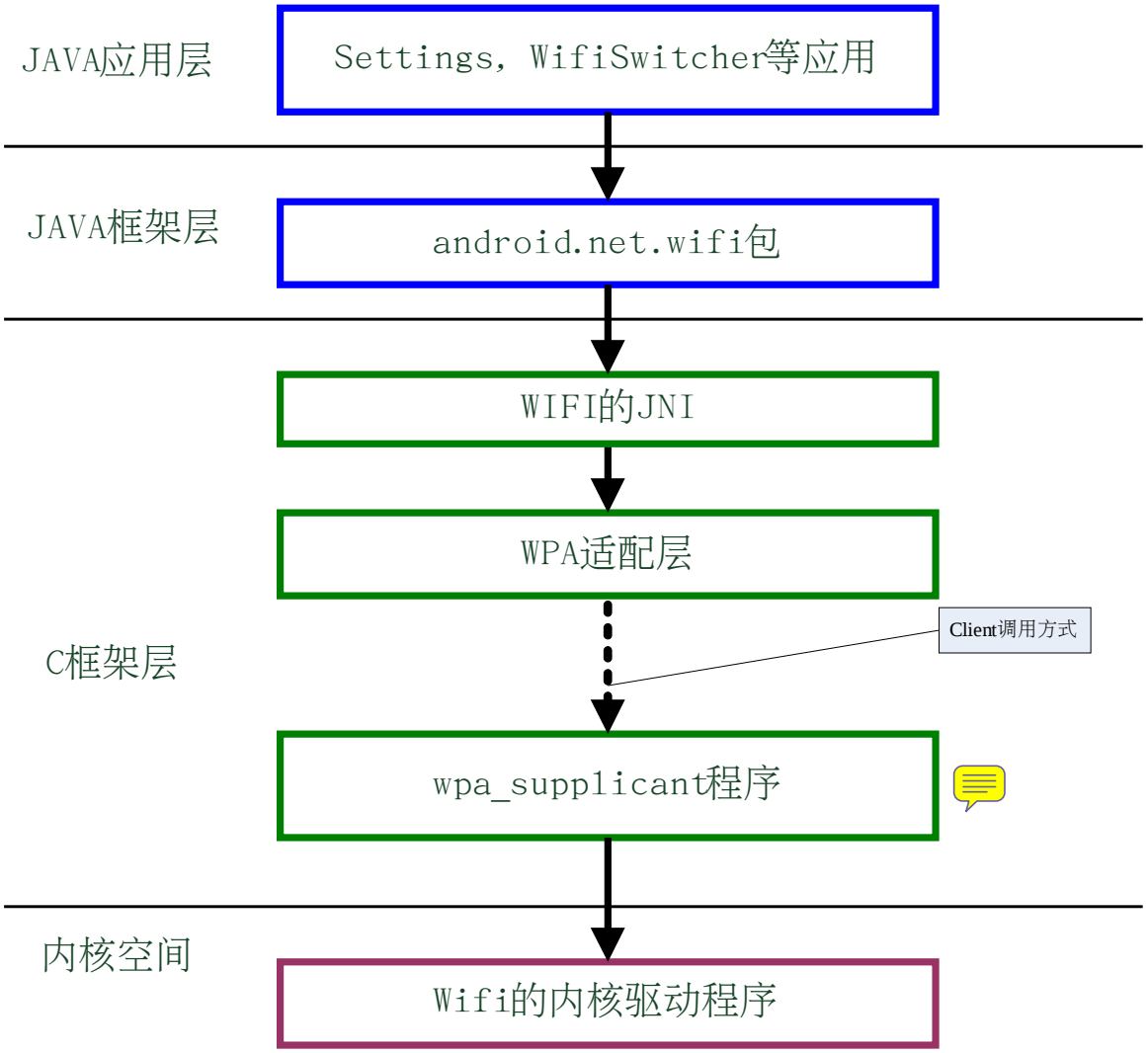
Wifi 用户空间的程序和库：
external/wpa_supplicant
生成库： libwpaclient.so 生成守护进程：
wpa_supplicant

Wifi 管理库：
hardware/libhardware_legacy/wifi/

Android 的 WIFI 系统的 JNI 的部分：
frameworks/base/core/jni/android_net_wifi_Wifi.cpp

WIFI 系统的的 JAVA 部分实现代码：
frameworks/base/services/java/com/android/server/
frameworks/base/wifi/java/android/net/wifi/

1.1 WIFI 的基本架构



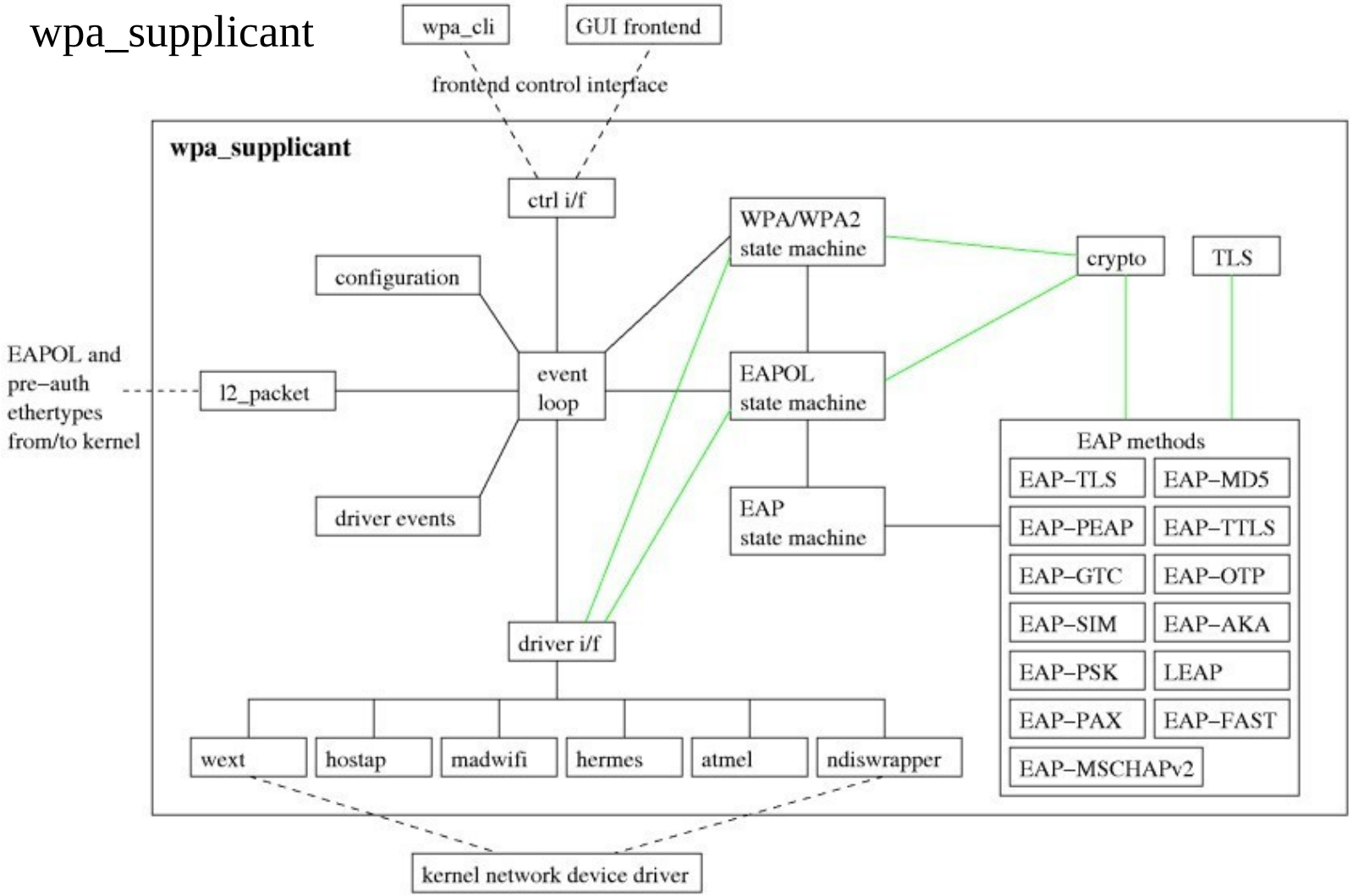
1.2 WIFI 的本地实现

Android 的 WiFi 本地实现部分主要包括 wpa_supplicant 以及 wpa_supplicant 适配层。

WPA 是 WiFi Protected Access 的缩写，中文含义为“WiFi 网络安全存取”。WPA 是一种基于标准的可互操作的 WLAN 安全性增强解决方案，可大大增强现有以及未来无线局域网系统的数据保护和访问控制水平。

1.2 WIFI 的本地实现

wpa_supplicant



1.2 WIFI 的本地实现

wpa_supplicant 适配层是通用的 wpa_supplicant 的封装，在 Android 中作为 WIFI 部分的硬件抽象层来使用。wpa_supplicant 适配层主要用于封装与 wpa_supplicant 守护进程的通信，以提供给 Android 框架使用。它实现了加载，控制和消息监控等功能。

wpa_supplicant 适配层的头文件如下所示：

hardware/libhardware_legacy/include/hardware_legacy/wifi.h

1.3 WIFI 的 JNI 和 JAVA 层次

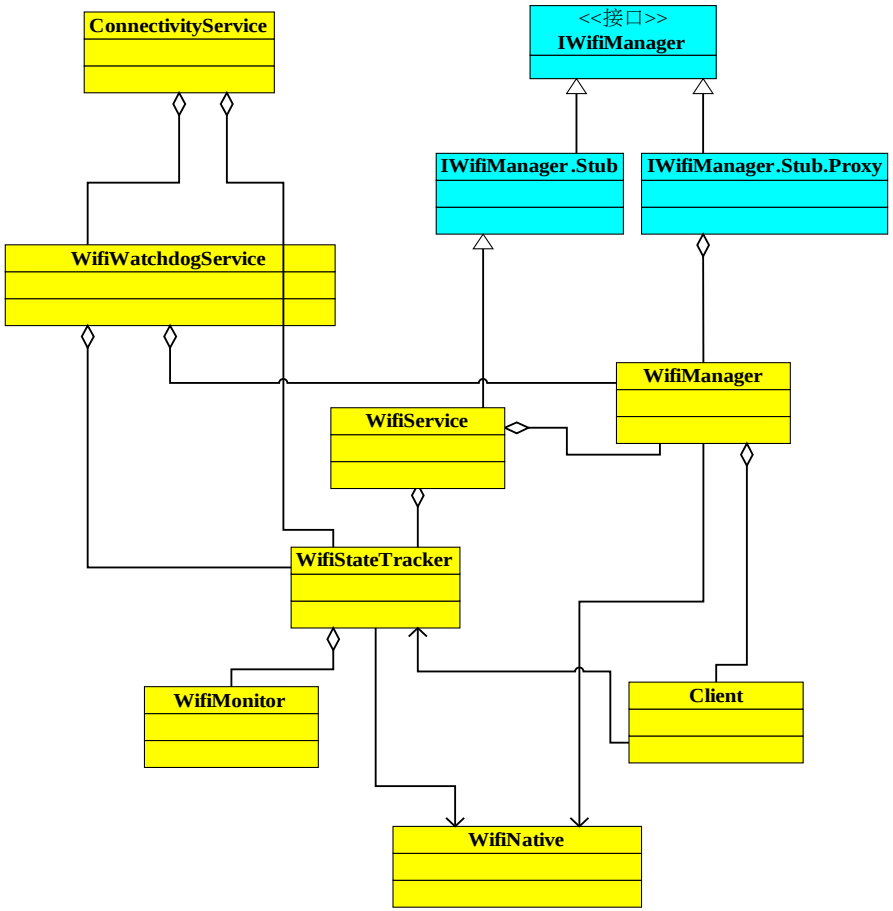
wpa_supplicant 适配层是通用的 wpa_supplicant 的封装，在 Android 中作为 WIFI 部分的硬件抽象层来使用。wpa_supplicant 适配层主要用于封装与 wpa_supplicant 守护进程的通信，以提供给 Android 框架使用。它实现了加载，控制和消息监控等功能。

wpa_supplicant 适配层的头文件如下所示:

hardware/libhardware_legacy/include/hardware_legacy/wifi.h

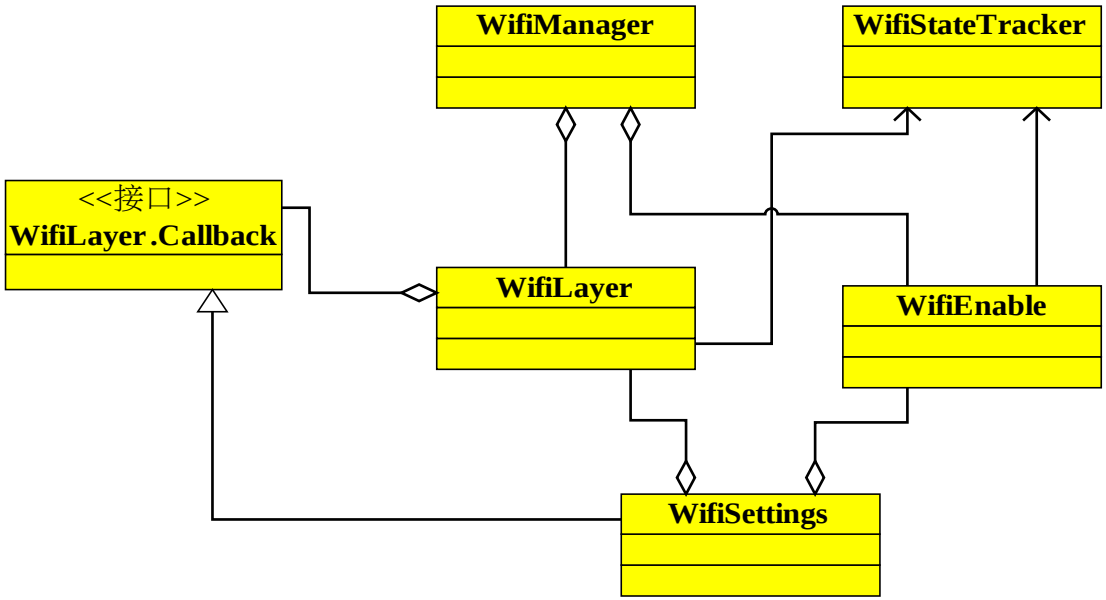
1.3 WIFI 的 JNI 和 JAVA 层次

WiFi 系统 Java 部分的核心是根据 IWifiManager 接口所创建的 Binder 服务器端和客户端，服务器端是 WifiService ， 客户端是 WifiManager 。



1.4 Settings 中的 WiFi 设置

Android 的 Settings 应用程序对 WiFi 的使用，是典型的 WiFi 应用方式，也是用户可见的 Android WiFi 管理程序。
这部分的实现代码在以下的目录中：
packages/apps/Settings/src/com/android/settings/wifi/
Settings 里的 WiFi 部分是用户可见的设置界面，提供 WiFi 开关、扫描 AP 、连接 / 断开等基本功能。另外，通过实现 WifiLayer.Callback 接口提供了一组回调函数，用以响应用户关心的 WiFi 状态的变化。



1.5 WiFi 工作流程实例

1．开启 WiFi

用户在设置界面开启 WiFi，调用 Settings 应用程序的 WifiEnabler.setWifiEnabled，然后调用 WifiManager.setWifiEnabled。

- WifiManager.setWifiEnabled 通过 Binder 机制调用 WifiService.setWifiEnabled。这里的 WifiService 也就是 WiFi 的 Java 层的内容。
- WifiService.setWifiEnabled 将 MESSAGE_ENABLE_WIFI 消息发送到自己的消息队列。

1.5 WiFi 工作流程实例

- WifiService 通过 WifiHandler 的 handleMessage 处理 MESSAGE_ENABLE_WIFI 。另外，它还完成一些初始工作，如设置当前状态、加载 WiFi 驱动、开启 wpa_supplicant 、开启 WifiStateTracker 、注册 BroadcastReceiver 监视 WifiStateTracker 的消息等。
- 由于 WifiEnabler 初始化时注册了 BroadcastReceiver ，因此它会获得这个通知消息，进入 handleWifiStateChanged 处理一些内部状态以及显示。
- WifiLayer 也同样获得了这个通知消息，至此， WiFi 开启完成。随后它的做法是查找 AP 。

1.5 WiFi 工作流程实例

2．查找 AP

WiFi 查找 AP 的过程如下所示：

- Settings 应用程序的 WifiLayer.attemptScan 调用 WifiManager.startScan 。
- Settings 应用程序的 WifiManager.startScan 通过 Binder 机制调用 WifiService.startScan 。
- WiFi 服务层的 WifiServiceWifiNative.scanCommand 通过 WifiNative 发送扫描命令给 wpa_supplicant，中间经过 JNI 实现中的 doCommand，最终调用 wap_supplicant 适配层的 wifi_command 来完成这一发送过程。至此，命令发送成功。

1.5 WiFi 工作流程实例

- 命令的最终响应由 wap_supplicant 上报“ SCAN-RESULTS” 消息， WifiStateTracker 开启的 WifiMonitor 的 MonitorThread 可以获取此消息并交由 handleEvent 处理。
- handleEvent 的处理方式是调用 WifiStateTracker.notifyScanResultsAvailable 。
- 在 WifiStateTracker 中，通过 EVENT_SCAN_RESULTS_AVAILABLE 完成消息传递，调用 sendScanResultsAvailable 将 SCAN_RESULTS_AVAILABLE_ACTION 通知消息广播出去。

WifiLayer 会最终获得这个通知消息，调用 handleScanResultsAvailable 继续处理。此函数会根据返回的 AP 数据建立对应的处理结构，并完成对应界面的绘制，以供用户操作 AP 列表。至此， AP 查找完成，也完成了一次典型的自上而下、再自下而上的情景。

1.5 WiFi 工作流程实例

3．连接 AP

WiFi 连接 AP 的步骤如下所示：

- 单击 AP 列表的某个项目后，会弹出 AccessPointDialog 对话框，单击“连接”按钮，将 handleConnect 转化为到 WifiLayer.connectToNetwork 的调用。
- 在 connectToNetwork 中完成一些查找和配置，再通过 managerEnableNetwork 调用 WifiManager.enableNetwork 。
- 连接的中间流程与查找 AP 的流程类似，都经过了 WifiMonitor 对“ CONNECTED” 消息响应的捕获，以及 WifiStateTracker 对 EVENT_SUPPLICANT_STATE_ CHANGED 的处理。还有一个比较重要的步骤是 WifiStateTracker 通过对 DHCP 服务器的申请进行了 IP 地址分配。最终会广播 NETWORK_STATE_CHANGED_ ACTION 消息，由 WifiLayer 响应。

第二部分 蓝牙部分

2.1 蓝牙部分的结构

2.2 Bluez

2.3 Bluez 的适配层

2.3 蓝牙的 JNI 和 JAVA 部分

2.1 蓝牙部分的结构

蓝牙(Bluetooth)技术，实际上是一种短距离无线电技术。

在Android中蓝牙除了kernel的支持，还需要用户空间的bluez的支持。

external/bluez/

Android 蓝牙设备管理的库:

system/bluetooth/

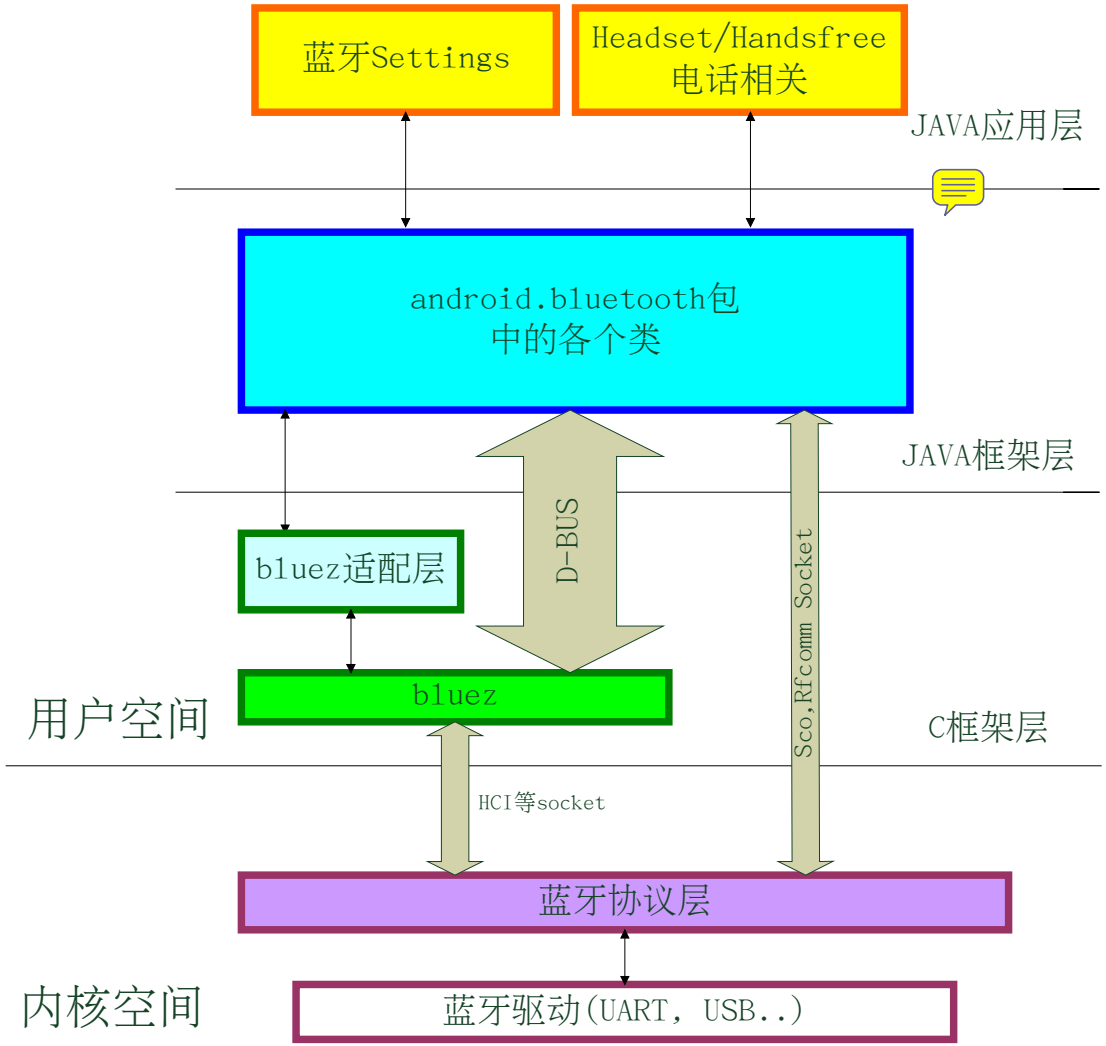
分别生成libbluetooth.so和libbluedroid.so。

2.4 蓝牙的 JNI 和 JAVA 部分

Bluetooth 的 JNI 到上层的接口，目录 frameworks/base/core/jni/ 中的 android_bluetooth_*.cpp

Bluetooth 的 JAVA 类：frameworks/base/core/java/android/bluetooth

2.1 蓝牙部分的结构

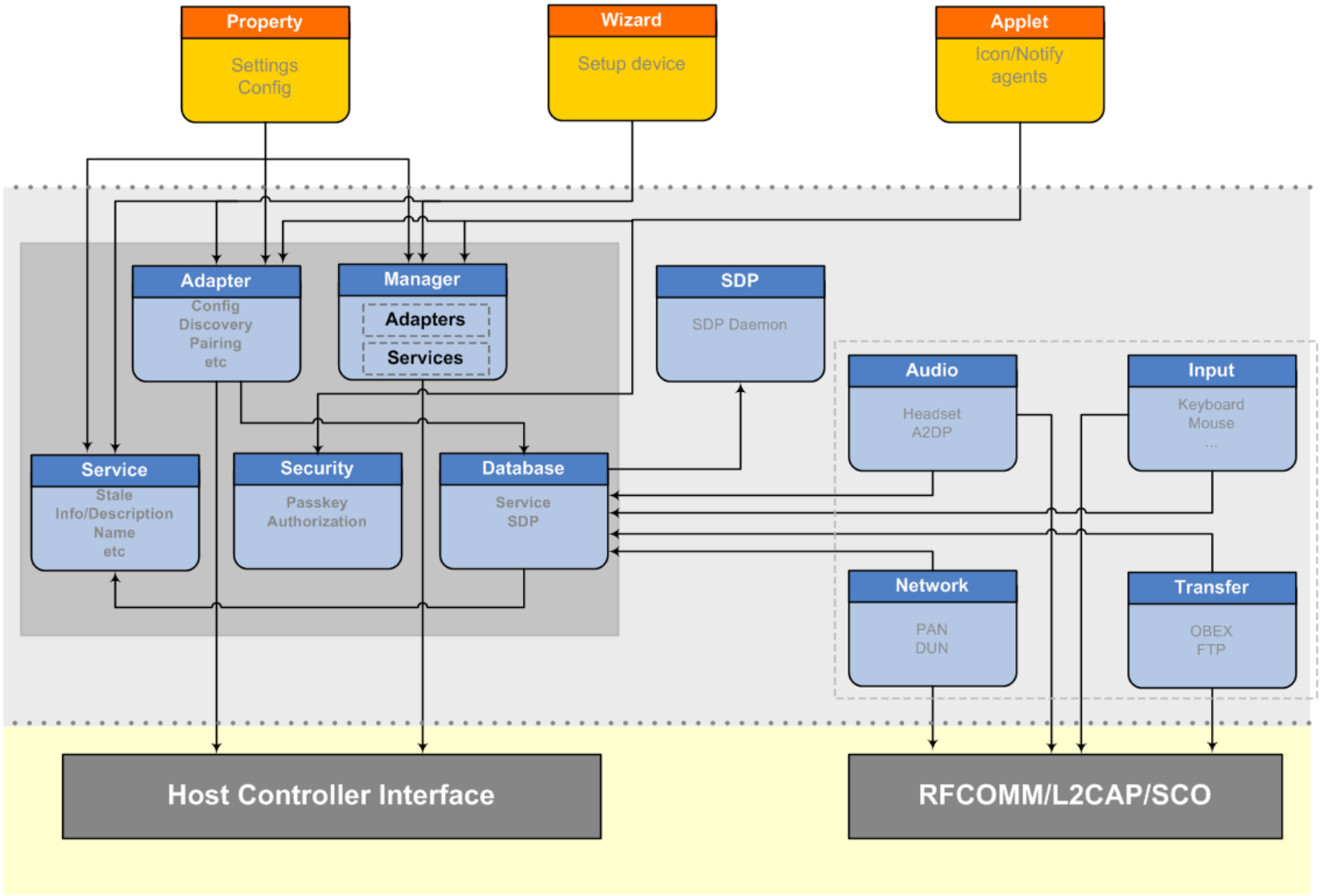

蓝牙Settings
Headset/Handsfree
电话相关
JAVA应用层
android.bluetooth包
中的各个类
JAVA框架层
D-BUS
Sco,Rfcomm Socket
bluez适配层
bluez
用户空间
C框架层
HCI等socket
蓝牙协议层
内核空间
蓝牙驱动(UART, USB..)

2.2 Bluez

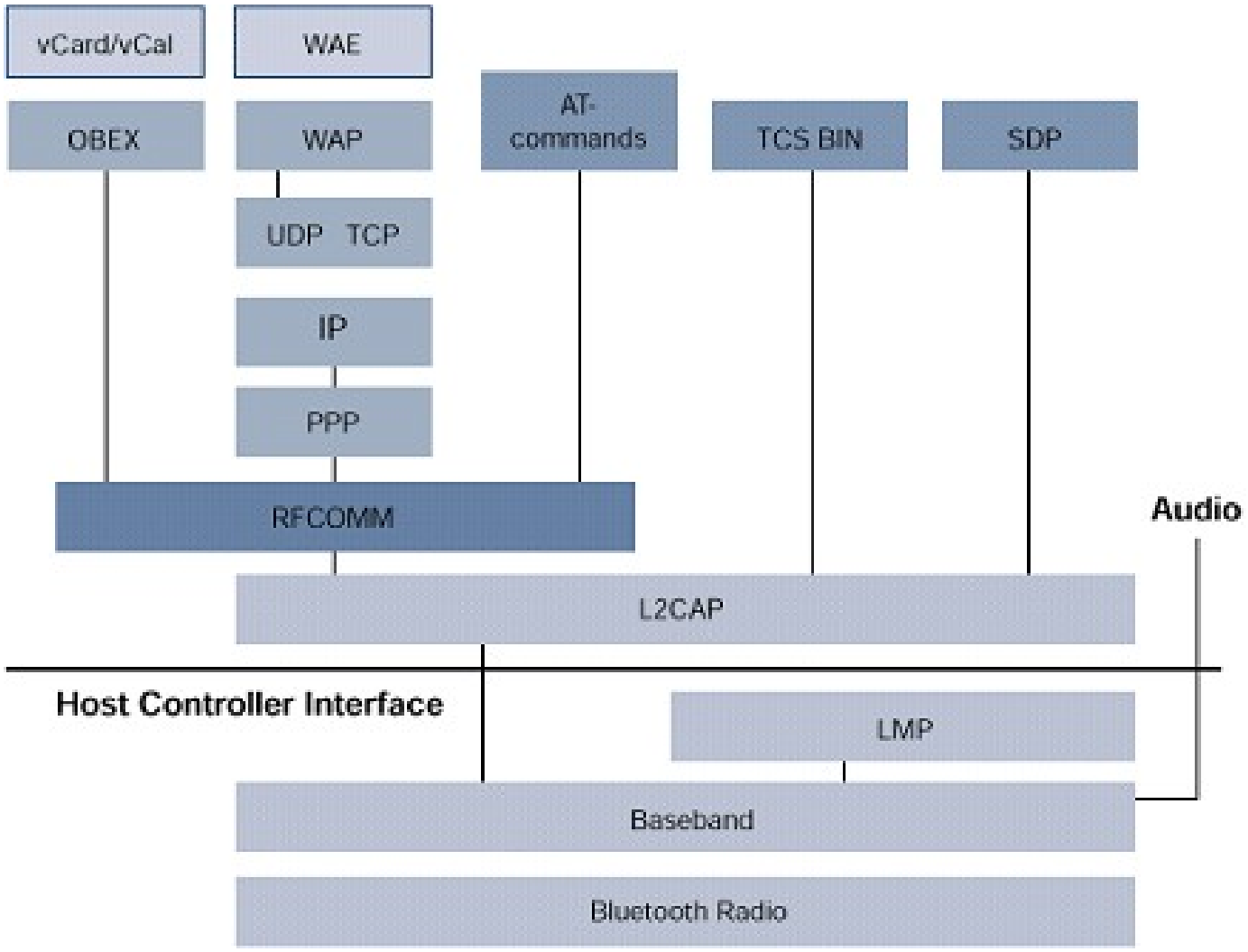
Android 所采用的蓝牙用库空间的库是 bluez 。它是一套 Linux 平台的蓝牙协议栈完整开源实现，广泛用在各 linux 发行版，并被移植到众多移动平台上。在 Android 中， bluez 提供了很多分散的应用，包括守护进程和一些工具。

bluez 通过 D-BUS IPC 机制来提供应用层接口。

2.2 Bluez


Property
Settings
Config
Wizard
Setup device
Applet
Icon/Notify
agents
Adapter
Config
Discovery
Pairing
etc
Manager
Adapters
Services
SDP
SDP Daemon
Audio
Headset
A2DP
Input
Keyboard
Mouse
...
Service
Stale
Info/Description
Name
etc
Security
Passkey
Authorization
Database
Service
SDP
Network
PAN
DUN
Transfer
OBEX
FTP
Host Controller Interface
RFCOMM/L2CAP/SCO

2.2 Bluez


vCard/vCal
WAE
OBEX
WAP
AT-commands
TCS BIN
SDP
UDP TCP
IP
PPP
RFCOMM
Audio
L2CAP
Host Controller Interface
LMP
Baseband
Bluetooth Radio

2.3 Bluez 的适配层

bluez 在 Android 中使用，需要经过 Android 的 bluez 适配层的封装， bluez 适配层源代码及头文件路径如下所示:

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，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。

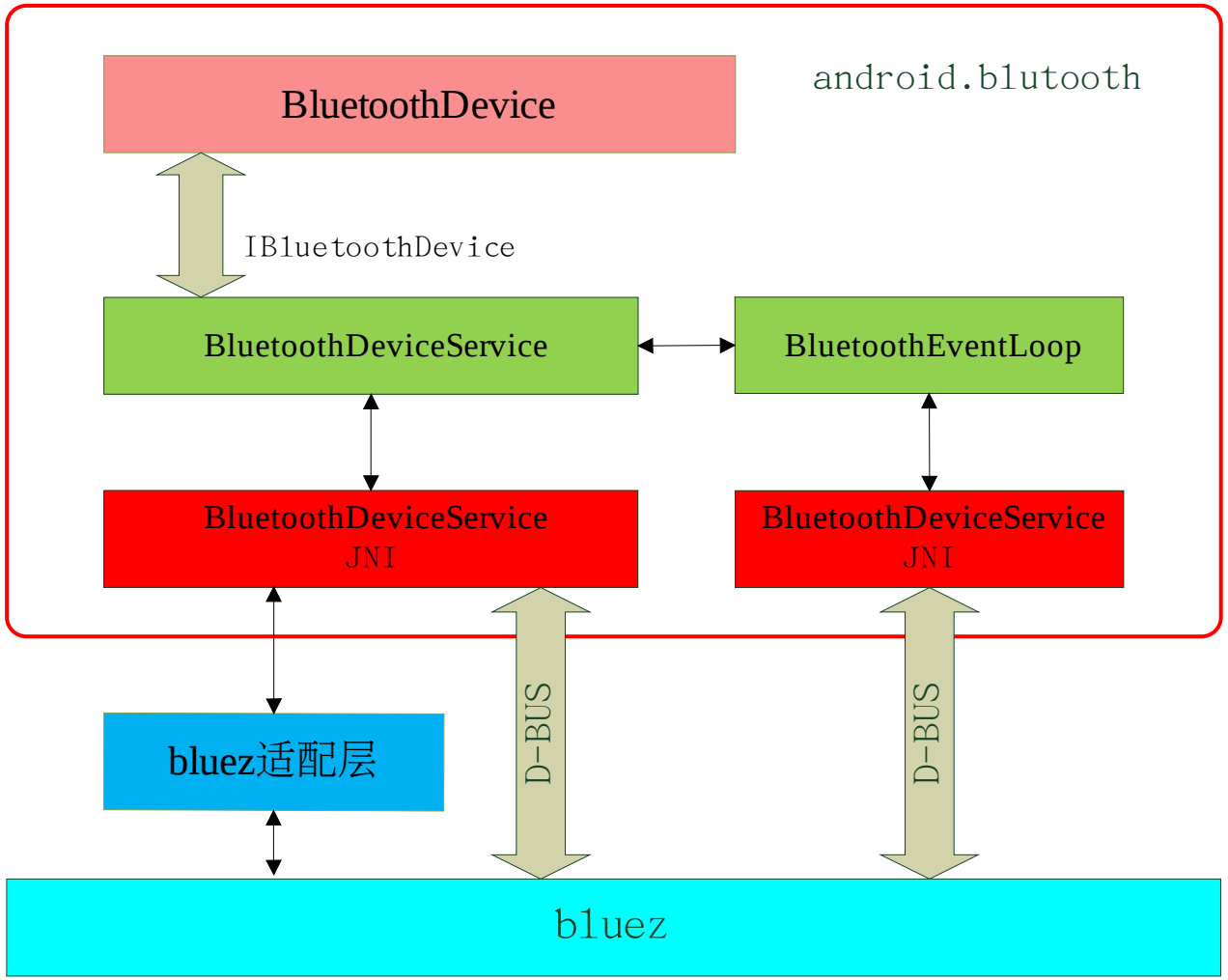
2.4 蓝牙的 JNI 和 JAVA 部分

在 Android 中还定义了 Bluetooth 通过 JNI 到上层的接口，在目录 frameworks/base/core/jni/ 中。

android_bluetooth_BluetoothAudioGateway.cpp
android_bluetooth_common.cpp
android_bluetooth_Database.cpp
android_bluetooth_ScoSocket.cpp
android_bluetooth_RfcommSocket.cpp
android_bluetooth_HeadsetBase.cpp

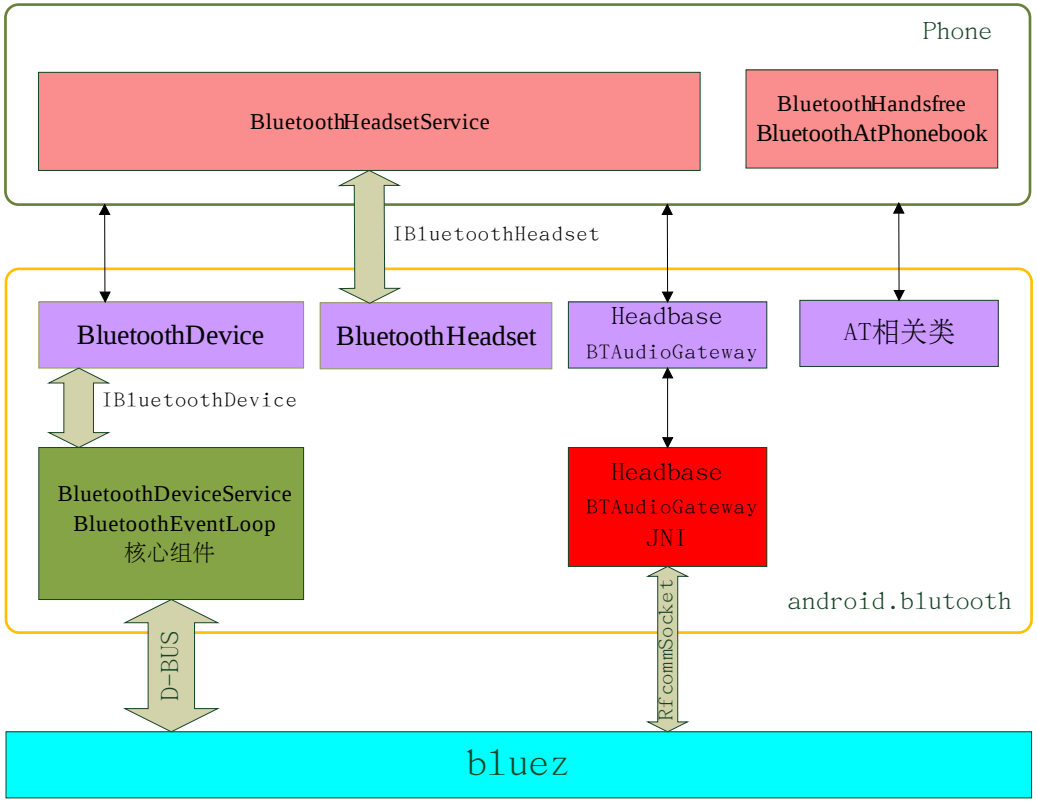
Bluetooth 的 JAVA 类：
frameworks/base/core/java/android/bluetooth

2.4 蓝牙的 JNI 和 JAVA 部分



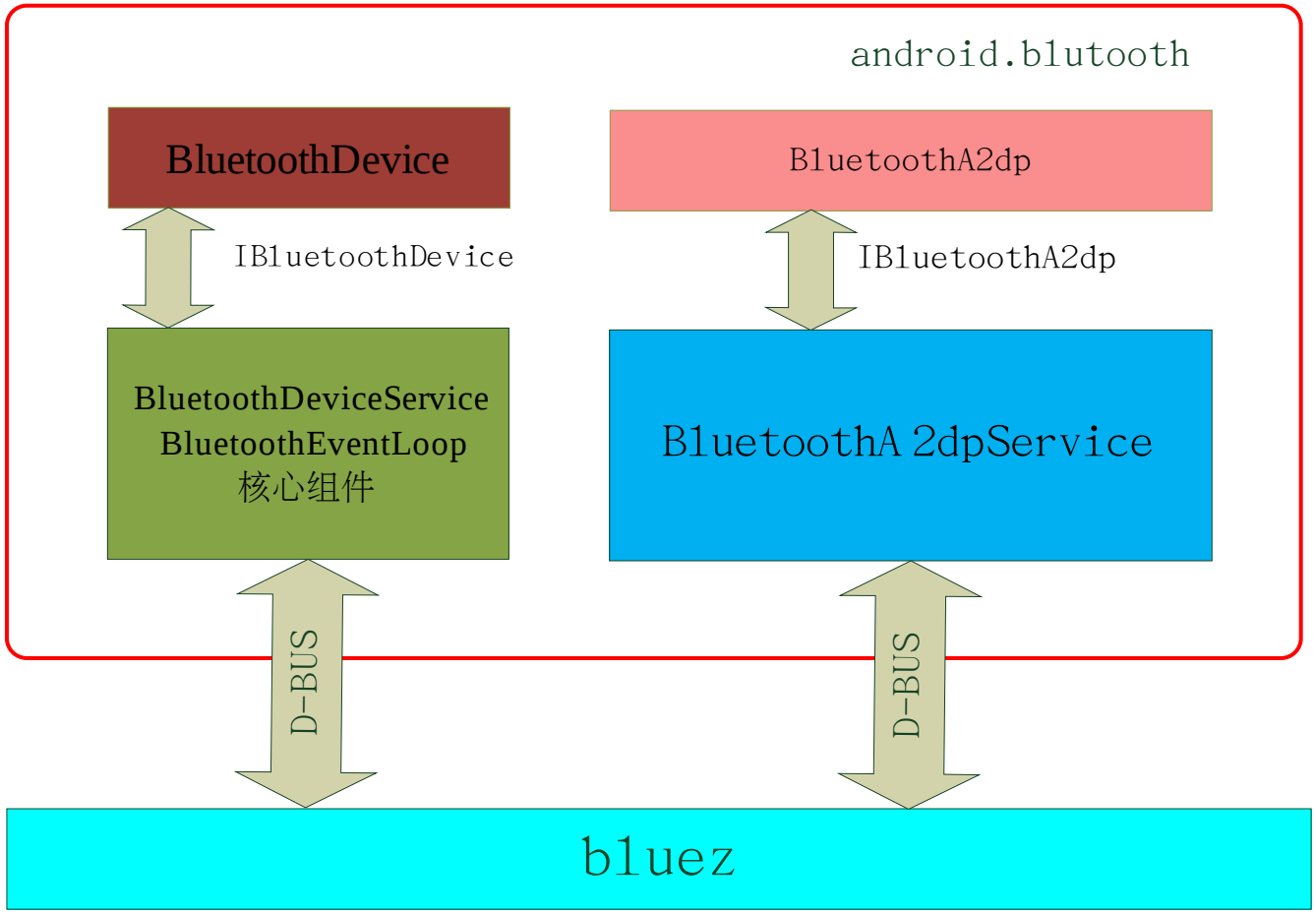
2.4 蓝牙的 JNI 和 JAVA 部分

耳机及免提服务



2.4 蓝牙的 JNI 和 JAVA 部分

A2DP 服务



第三部分　GPS 和定位部分

在目前的手机平台上，定位系统有着十分重要的地位， Android 也是如此。定位系统的最常见设备如 GPS（Global Position System, 全球定位系统），另外还包括利用基站（Cell）定位的 AGPS 等设备。它们在提供基本定位功能的同时，也为很多新奇的应用提供了基础。

第三部分　GPS 和定位部分

3.1　定位系统基本架构

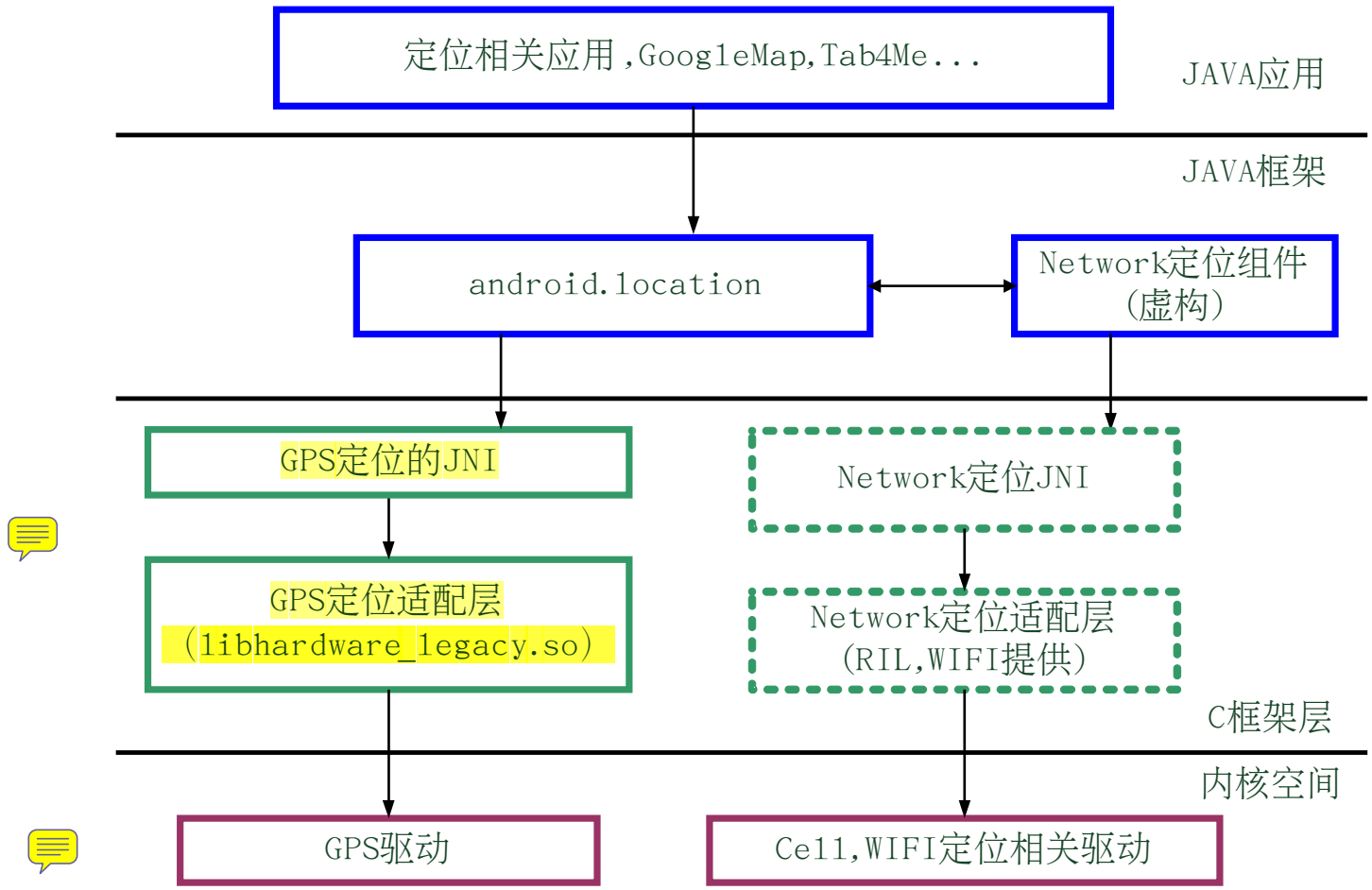
3.2　定位系统驱动层

3.3　GPS 本地实现

3.4　GPS　JNI 实现

3.5　定位系统 Java 实现

3.1 定位系统基本架构


定位相关应用,GoogleMap,Tab4Me...
JAVA应用
JAVA框架
android.location
Network定位组件
(虚构)
GPS定位的JNI
Network定位JNI
GPS定位适配层
(libhardware_legacy.so)
Network定位适配层
(RIL,WIFI提供)
C框架层
内核空间
GPS驱动
Cell,WIFI定位相关驱动

3.1 定位系统基本架构

Android 在定位系统方面有着比较系统的架构，让各种定位设备可以方便地集成进来，也让基于定位的应用开发变得更加容易。

Android 定位系统的主要数据来源有两个，分别是 GPS 定位和 Network 定位（基于 Cell 或 WiFi 热点的定位）。在 Network 定位中， Cell 定位已经相当常见， WiFi 热点定位现在即使在国外也不是很普及。

对 Android 的源码来说， **GPS 相关部分是开源项目的一部分，而 Network 定位部分只在开源代码中提供了接口**。两者虽然底层技术实现不同，但作为定位数据的提供方都有着很多共同的地方，并使用同一套框架。

3.1 定位系统基本架构

GPS 本地实现部分主要是 GPS 驱动适配层，头文件路径为：
hardware/libhardware_legacy/include/hardware_legacy/gps.h
GPS 部分的源代码在以下的目录中：
hardware/libhardware_legacy/gps/

GPS 部分的 JNI 的本地部分实现源码，为以下路径：
frameworks/base/core/jni/android_location_GpsLocationProvider.cpp

GPS 部分的 JAVA 层的实现代码在：
frameworks/base/location/java/android/location

3.2 定位系统驱动层

GPS 本地实现部分主要是 GPS 驱动适配层，头文件路径为：
hardware/libhardware_legacy/include/hardware_legacy/gps.h
GPS 部分的源代码在以下的目录中：
hardware/libhardware_legacy/gps/

GpsInterface 的内容：

```
typedef struct {                /*  定义标准的 GPS 接口的结构体  */
    int  (*init)( GpsCallbacks* callbacks );    /*  初始化 GPS，提供回调函数实现  */
    int  (*start)( void );                      /*  开始导航  */
    int  (*stop)( void );                       /*  停止导航  */
    void (*set_fix_frequency)( int frequency ); /*  设置请求频率（定点）  */
    void (*cleanup)( void );                    /*  关闭接口  */
    int  (*inject_time)(GpsUtcTime time,        /*  置入当前的时间  */
                        int64_t timeReference, int uncertainty);
    void (*delete_aiding_data)(GpsAidingData flags);/*  删除帮助信息  */
    int  (*set_position_mode)(GpsPositionMode mode,
                              int fix_frequency);   /*  设置位置模式  */
    const void* (*get_extension)(const char* name);  /*  获得扩展信息的指针  */
} GpsInterface;
```

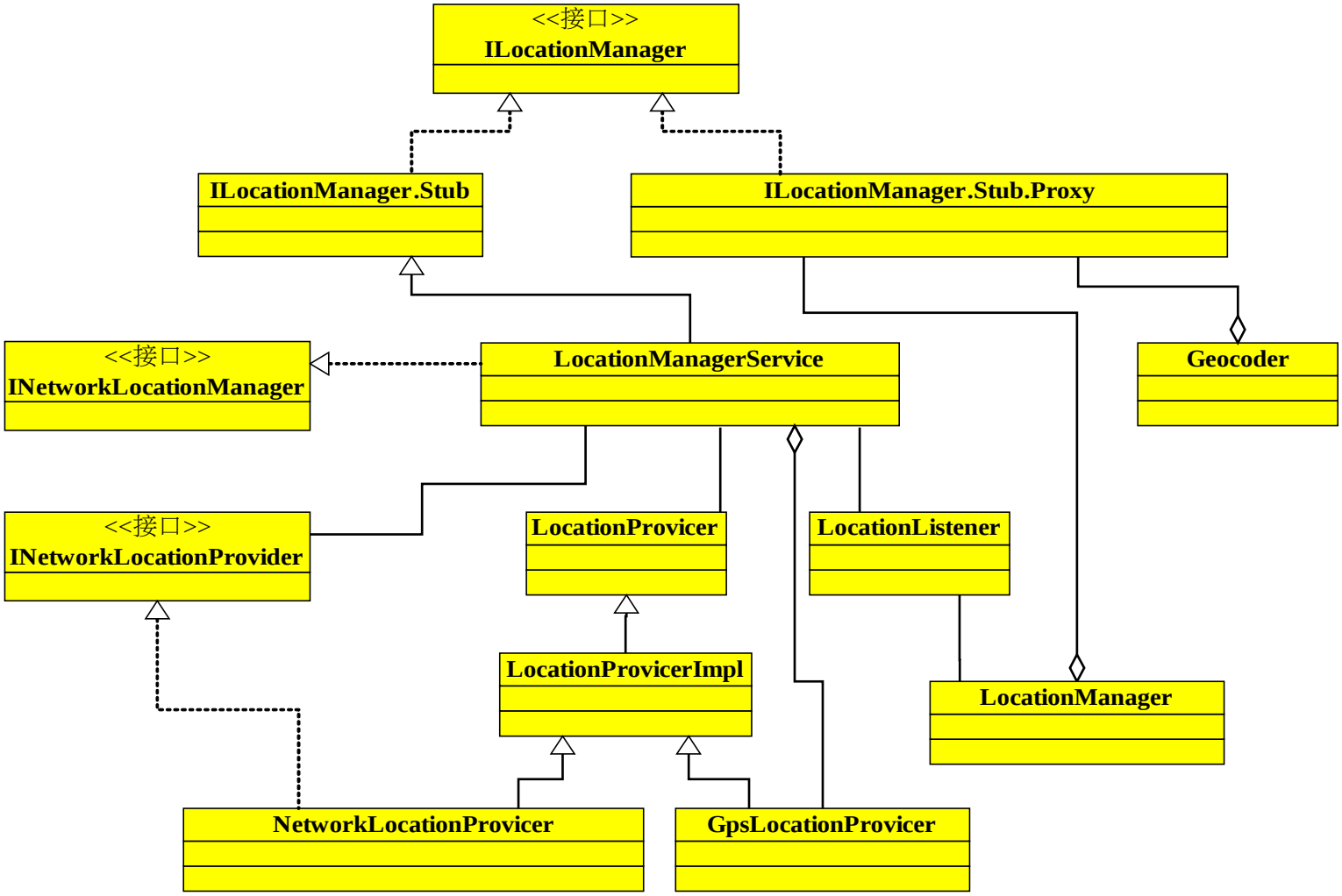
3.3 GPS 本地实现

GPS 硬件设备分为**硬 GPS 和软 GPS** 。硬 GPS 一般是功能独立的模块，一般不需要特别多的控制，上电就可以运行，直接输出 NMEA 数据，驱动十分简单。软 GPS 一般需要主控芯片控制其运行状态，输出的大多是裸卫星数据，需要主控方进行计算，才能得到最终的 NMEA 数据。两者的共同点是，最终的输出都是 NMEA 数据。

NMEA （National Marine Electronics Association，国际海洋电子协会）一般是指 NMEA0183，这是一套工业标准的接收机信号输出协议。

http://www.nmea.org/

第三部分　GPS 和定位部分



谢谢！