Android 蓝牙开发技术

要学习蓝牙先认识

一 RFCOMM 通道：

RFCOMM 协议

一个基于欧洲电信标准协会 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，其目的是针对如何在两个不同设备上的应用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。

RFCOMM 协议概述

RFCOMM 通信段

RFCOMM 是为了兼容传统的串口应用，同时取代有线的通信方式，蓝牙协议栈需要提供与有线串口一致的通信接口而开发出的协议。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。

目的：

在两个不同设备(通信设备的两端)上的应用之间保证一条完整的通信路径,并在他们之间保持一通信段。下图是一条完整的通信路径。

RFCOMM 只针对直接互连设备之间的连接，或者是设备与网络接入设备之间的互连。通信两端设备必须兼容于 RFCOMM 协议，有两类设备：DTE (Data Terminal Endpoint，通信终端，如 PC，PRINTER)和 DCE (Data Circuit Endpoint，通信段的一部分，如 Modem)。此两类设备不作区分。

RFCOMM 服务

RFCOMM 仿真 RS232 串口，仿真过程包括非数据通路状态的传输，RFCOMM 内置空 Modem 仿真标准框架。

RFCOMM 中的仿真 RS-232 通路

多串口仿真

两个采用 RFCOMM 通信的 BT 设备有可能同时打开多个串口，RFCOMM 支持同时打开 60 个端口。

认识二：MAC 硬件地址

MAC(Medium/MediaAccess Control, 介质访问控制)MAC 地址是烧录在 NetworkInterfaceCard(网卡,NIC)里的.MAC 地址,也叫硬件地址,是由 48 比特长(6 字节),16 进制的数字组成.0-23 位叫做组织唯一标志符(organizationally unique，是识别 LAN(局域网)节点的标识.24-47 位是由厂家自己分配。其中第 40 位是组播地址标志位。网卡的物理地址通常是由网卡生产厂家烧入网卡的 EPROM(一种闪存芯片，通常可以通过程序擦写)，它存储的是传输数据时真正赖以标识发出数据的电脑和接收数据的主机的地址。

也就是说，在网络底层的物理传输过程中，是通过物理地址来识别主机的，它一般也是全球唯一的。比如，著名的以太网卡，其物理地址是 48bit(比特位)的整数，如：44-45-53-54-00-00,以机器可读的方式存入主机接口中。以太网地址管理机构(除了管这个外还管别的)

(IEEE)(IEEE：电气和电子工程师协会)将以太网地址，也就是 48 比特的不同组合，分为若干独立的连续地址组，生产以太网网卡的厂家就购买其中一组，具体生产时，逐个将唯一地址赋予以太网卡。

形象的说，MAC 地址就如同我们身份证上的身份证号码，具有全球唯一性。

步骤一：Setting Up Bluetooth

通过 BluetoothAdapter 得到蓝牙的 Activity

发送蓝牙连接意图

通过 e onActivityResult()得到蓝牙连接意图

步骤二： Finding Devices

通过得到开启蓝牙用户名和 MAC 地址

配对蓝牙

步骤三：连接蓝牙

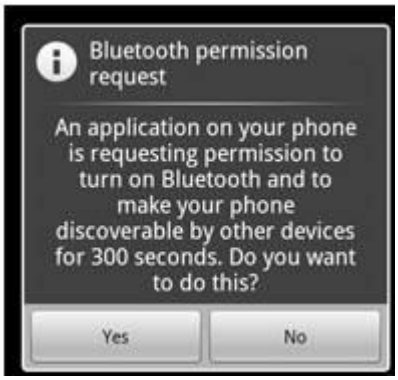
就像 java 的聊天系统一样用一个蓝牙手机当服务器，一个当客户端，在用一个类当做连接的管理类就行了

android 平台蓝牙编程

Android 平台支持蓝牙网络协议栈，实现蓝牙设备之间数据的无线传输。

本文档描述了怎样利用 android 平台提供的蓝牙 API 去实现蓝牙设备之间的通信，蓝牙设备之间的通信主要包括了四个步骤：设置蓝牙设备、寻找局域网内可能或者匹配的设备、连接设备和设备之间的数据传输。以下是建立蓝牙连接的所需要的一些基本类：

BluetoothAdapter 类：代表了一个本地的蓝牙适配器。他是所有蓝牙交互的的入口点。利用它你可以发现其他蓝牙设备，查询绑定了的设备，使用已知的 MAC 地址实例化一个蓝牙设备和建立一个 BluetoothServerSocket（作为服务器端）来监听来自其他设备的连接。



BluetoothDevice 类：代表了一个远端的蓝牙设备，使用它请求远端蓝牙设备连接或者获取远端蓝牙设备的名称、地址、种类和绑定状态。（其信息是封装在 bluetoothsocket 中）。

Bluetoothsocket 类：代表了一个蓝牙套接字的接口（类似于 tcp 中的套接字），他是应用程序通过输入、输出流与其他蓝牙设备通信的连接点。

Blueboothserversocket 类：代表打开服务连接来监听可能到来的连接请求（属于 server 端），为了连接两个蓝牙设备必须有一个设备作为服务器打开一个服务套接字。当远端设备发起连接连接请求的时候，并且已经连接到了的时候，Blueboothserversocket 类将会返回一个 bluetoothsocket。

Bluetoothclass 类：描述了一个蓝牙设备的一般特点和能力。他的只读属性集定义了设备的主、次设备类和一些相关服务。然而，他并没有准确的描述所有该设备所支持的蓝牙文件和服务，而是作为对设备种类来说的一个小小暗示。

下面说说具体的编程实现：

必须确定你的设备支持蓝牙，并保证他可以用。如果你的设备支持蓝牙，将它使能。当然，有两种方法，一种是在你的系统设置里开启蓝牙，另外一中是在你的应用程序里启动蓝牙功能，第一种方法就不讲了，具体讲一个第二种方法：

首先通过调用静态方法 getDefaultAdapter()获取蓝牙适配器 bluetoothadapter，以后你就可以使用该对象了。如果返回为空，the story is over。

Eg：BluetoothAdapter mBluetoothAdapter = BluetoothAdapter.getDefaultAdapter();

```
if (mBluetoothAdapter == null) {
    // Device does not support Bluetooth
}
```

其次，调用 isEnabled()来查询当前蓝牙设备的状态，如果返回为 false，则表示蓝牙设备没有开启，接下来你需要封装一个 ACTION_REQUEST_ENABLE 请求到 intent 里面，调用 startActivityForResult()方法使能蓝牙设备，例如：

```
if (!mBluetoothAdapter.isEnabled()) {
        Intent enableBtIntent = new Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
```

```
    startActivityForResult(enableBtIntent, REQUEST_ENABLE_BT);
}
```

至此，如不出意外，恭喜你的蓝牙设备已经开启了，接下来需要查找周边可能存在的蓝牙设备了。

查找设备：

使用 bluetoothadapter 类里的方法，你可以查找远端设备（不过蓝牙查找的范围好像是在十米以内吧）或者查询在你手机上已经匹配（或者说绑定）的其他手机了。当然需要确定对方蓝牙设备已经开启或者已经开启了“被发现使能“功能（对方设备是可以被发现的是你能够发起连接的前提条件）。如果该设备是可以被发现的，会反馈回来一些对方的设备信息，比如名字、MAC 地址等，利用这些信息，你的设备就可以选择去向对方初始化一个连接。如果你是第一次与该设备连接，那么一个配对的请求就会自动的显示给用户。当设备配对好之后，他的一些基本信息（主要是名字和 MAC）被保存下来并可以使用蓝牙的 API 来读取。使用已知的 MAC 地址就可以对远端的蓝牙设备发起连接请求。

匹配好的设备和连接上的设备的不同点：匹配好只是说明对方设备发现了你的存在，并拥有一个共同的识别码，并且可以连接。连接上：表示当前设备共享一个 RFCOMM 信道并且两者之间可以交换数据。也就是是说蓝牙设备在建立 RFCOMM 信道之前，必须是已经配对好了的。

怎么查询匹配好的设备：

在建立连接之前你必须先查询配对好了的蓝牙设备集（你周围的蓝牙设备可能不止一个），以便你选取哪一个设备进行通信，例如你可以你可以查询所有配对的蓝牙设备，并使用一个数组适配器将其打印显示出来：

```
   Set<BluetoothDevice> pairedDevices = mBluetoothAdapter.getBondedDevices();
// If there are paired devices
if (pairedDevices.size() > 0) {
    // Loop through paired devices
    for (BluetoothDevice device : pairedDevices) {
        // Add the name and address to an array adapter to show in a ListView
        mArrayAdapter.add(device.getName() + "\n" + device.getAddress());
    }
```

建立一个蓝牙连接只需要 MAC 地址就已经足够了。

扫描设备：

扫描设备，只需要简单的调用 startDiscovery()方法，这个扫描的过程大概持续是 12 秒，应用程序为了 ACTION_FOUND 动作需要注册一个 BroadcastReceiver 来接受设备扫描到的信息。对于每一个设备，系统都会广播 ACTION_FOUND 动作。例如：

```
    // Create a BroadcastReceiver for ACTION_FOUND
private final BroadcastReceiver mReceiver = new BroadcastReceiver() {
    public void onReceive(Context context, Intent intent) {
        String action = intent.getAction();
        // When discovery finds a device
        if (BluetoothDevice.ACTION_FOUND.equals(action)) {
```

```
                // Get the BluetoothDevice object from the Intent
                BluetoothDevice device = intent.getParcelableExtra(BluetoothDevice.EXTRA_DEVICE);
                // Add the name and address to an array adapter to show in a ListView
                mArrayAdapter.add(device.getName() + "\n" + device.getAddress());
            }
        }
};
// Register the BroadcastReceiver
IntentFilter filter = new IntentFilter(BluetoothDevice.ACTION_FOUND);
registerReceiver(mReceiver, filter); // Don't forget to unregister during onDestroy
```

注意：扫描的过程是一个很耗费资源的过程，一旦你找到你需要的设备之后，在发起连接请求之前，确保你的程序调用 cancelDiscovery()方法停止扫描。显然，如果你已经连接上一个设备，启动扫描会减少你的通信带宽。

使能被发现：Enabling discoverability

如果你想使你的设备能够被其他设备发现，将 ACTION_REQUEST_DISCOVERABLE 动作封装在 intent 中并调用 startActivityForResult(Intent, int)方法就可以了。他将在不使你应用程序退出的情况下使你的设备能够被发现。缺省情况下的使能时间是 120 秒，当然你可以可以通过添加 EXTRA_DISCOVERABLE_DURATION 字段来改变使能时间（最大不超过 300 秒，这是出于对你设备上的信息安全考虑）。例如：

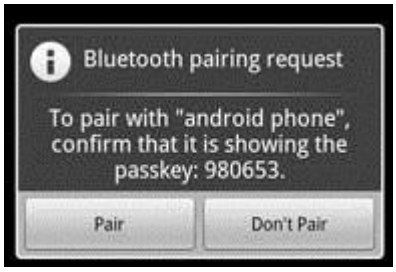
```
Intent discoverableIntent = new
Intent(BluetoothAdapter.ACTION_REQUEST_DISCOVERABLE);
discoverableIntent.putExtra(BluetoothAdapter.EXTRA_DISCOVERABLE_DURATION, 300);
startActivity(discoverableIntent);
```

运行该段代码之后，系统会弹出一个对话框来提示你启动设备使能被发现（次过程中如果你的蓝牙功能没有开启，系统会帮你开启），并且如果你准备对该远端设备发现一个连接，你不需要开启使能设备被发现功能，以为该功能只是在你的应用程序作为服务器端的时候才需要。

连接设备：

在你的应用程序中，想建立两个蓝牙设备之间的连接，你必须实现客户端和服务器端的代码（因为任何一个设备都必须可以作为服务端或者客户端）。一个开启服务来监听，一个发起连接请求（使用服务器端设备的 MAC 地址）。当他们都拥有一个蓝牙套接字在同一 RFECOMM 信道上的时候，可以认为他们之间已经连接上了。服务端和客户端通过不同的方式或其他们的蓝牙套接字。当一个连接监听到的时候，服务端获取到蓝牙套接字。当客户可打开一个 FRCOMM 信道给服务器端的时候，客户端获取到蓝牙套接字。

注意：在此过程中，如果两个蓝牙设备还没有配对好的，android 系统会通过一个通知或者对话框的形式来通知用户。RFCOMM 连接请求会在用户选择之前阻塞。如下图：



服务端的连接：

当你想要连接两台设备时，一个必须作为服务端（通过持有一个打开的 bluetoothserversocket），目的是监听外来连接请求，当监听到以后提供一个连接上的 bluetoothsocket 给客户端，当客户端从 bluetoothserversocket 得到 bluetoothsocket 以后就可以销毁 bluetoothserversocket，除非你还想监听更多的连接请求。

建立服务套接字和监听连接的基本步骤：

首先通过调用 listenUsingRfcommWithServiceRecord(String, UUID) 方法来获取 bluetoothserversocket 对象，参数 string 代表了该服务的名称，UUID 代表了和客户端连接的一个标识（128 位格式的字符串 ID，相当于 pin 码），UUID 必须双方匹配才可以建立连接。其次调用 accept（）方法来监听可能到来的连接请求，当监听到以后，返回一个连接上的蓝牙套接字 bluetoothsocket。最后，在监听到一个连接以后，需要调用 close（）方法来关闭监听程序。（一般蓝牙设备之间是点对点的传输）

注意：accept（）方法不应该放在主 Acitvity 里面，因为他是一种阻塞调用（在没有监听到连接请求之间程序就一直停在那里）。解决方法是新建一个线程来管理。例如：

```
 private class AcceptThread extends Thread {
    private final BluetoothServerSocket mmServerSocket;

    public AcceptThread() {
        // Use a temporary object that is later assigned to mmServerSocket,
        // because mmServerSocket is final
        BluetoothServerSocket tmp = null;
        try {
            // MY_UUID is the app's UUID string, also used by the client code
            tmp = mAdapter.listenUsingRfcommWithServiceRecord(NAME, MY_UUID);
        } catch (IOException e) { }
        mmServerSocket = tmp;
    }

    public void run() {
        BluetoothSocket socket = null;
        // Keep listening until exception occurs or a socket is returned
        while (true) {
            try {
                socket = mmServerSocket.accept();
            } catch (IOException e) {
                break;
            }
```

```
                // If a connection was accepted
                if (socket != null) {
                    // Do work to manage the connection (in a separate thread)
                    manageConnectedSocket(socket);
                    mmServerSocket.close();
                    break;
                }
            }
        }

        /** Will cancel the listening socket, and cause the thread to finish */
        public void cancel() {
            try {
                mmServerSocket.close();
            } catch (IOException e) { }
        }
}
```

客户端的连接：

为了初始化一个与远端设备的连接，需要先获取代表该设备的一个 bluetoothdevice 对象。通过 bluetoothdevice 对象来获取 bluetoothsocket 并初始化连接：

具体步骤：

使用 bluetoothdevice 对象里的方法 createRfcommSocketToServiceRecord(UUID)来获取 bluetoothsocket。UUID 就是匹配码。然后，调用 connect（）方法来。如果远端设备接收了该连接，他们将在通信过程中共享 RFFCOMM 信道，并且 connect（）方法返回。例如：

```
    private class ConnectThread extends Thread {
        private final BluetoothSocket mmSocket;
        private final BluetoothDevice mmDevice;

        public ConnectThread(BluetoothDevice device) {
            // Use a temporary object that is later assigned to mmSocket,
            // because mmSocket is final
            BluetoothSocket tmp = null;
            mmDevice = device;

            // Get a BluetoothSocket to connect with the given BluetoothDevice
            try {
                // MY_UUID is the app's UUID string, also used by the server code
                tmp = device.createRfcommSocketToServiceRecord(MY_UUID);
            } catch (IOException e) { }
            mmSocket = tmp;
        }

        public void run() {
```

```
        // Cancel discovery because it will slow down the connection
        mAdapter.cancelDiscovery();

        try {
            // Connect the device through the socket. This will block
            // until it succeeds or throws an exception
            mmSocket.connect();
        } catch (IOException connectException) {
            // Unable to connect; close the socket and get out
            try {
                mmSocket.close();
            } catch (IOException closeException) { }
            return;
        }

        // Do work to manage the connection (in a separate thread)
        manageConnectedSocket(mmSocket);
    }

    /** Will cancel an in-progress connection, and close the socket */
    public void cancel() {
        try {
            mmSocket.close();
        } catch (IOException e) { }
    }
```

注意：conncet（）方法也是阻塞调用，一般建立一个独立的线程中来调用该方法。在设备 discover 过程中不应该发起连接 connect（），这样会明显减慢速度以至于连接失败。且数据传输完成只有调用 close（）方法来关闭连接，这样可以节省系统内部资源。

管理连接（主要涉及数据的传输）：

当设备连接上以后，每个设备都拥有各自的 bluetoothsocket。现在你就可以实现设备之间数据的共享了。

1. 首先通过调用 getInputStream()和 getOutputStream()方法来获取输入输出流。然后通过调用 read(byte[]) 和 write(byte[]).方法来读取或者写数据。
2. 实现细节：以为读取和写操作都是阻塞调用，需要建立一个专用现成来管理。
3. private class ConnectedThread extends Thread {

```
    private final BluetoothSocket mmSocket;
    private final InputStream mmInStream;
    private final OutputStream mmOutStream;

    public ConnectedThread(BluetoothSocket socket) {
        mmSocket = socket;
        InputStream tmpIn = null;
        OutputStream tmpOut = null;
```

```
        // Get the input and output streams, using temp objects because
        // member streams are final
        try {
            tmpIn = socket.getInputStream();
            tmpOut = socket.getOutputStream();
        } catch (IOException e) { }

        mmInStream = tmpIn;
        mmOutStream = tmpOut;
    }

    public void run() {
        byte[] buffer = new byte[1024];    // buffer store for the stream
        int bytes; // bytes returned from read()

        // Keep listening to the InputStream until an exception occurs
        while (true) {
            try {
                // Read from the InputStream
                bytes = mmInStream.read(buffer);
                // Send the obtained bytes to the UI Activity
                mHandler.obtainMessage(MESSAGE_READ, bytes, -1, buffer)
                        .sendToTarget();
            } catch (IOException e) {
                break;
            }
        }
    }

    /* Call this from the main Activity to send data to the remote device */
    public void write(byte[] bytes) {
        try {
            mmOutStream.write(bytes);
        } catch (IOException e) { }
    }

    /* Call this from the main Activity to shutdown the connection */
    public void cancel() {
        try {
            mmSocket.close();
        } catch (IOException e) { }
    }
}
```