**1.简介**

关于蓝牙的介绍对于阅读本文章的人来说并不需要我进行对于的介绍。在Android中为蓝牙的使用提供了一整套API，可以进行如下操作：

* 搜索其他蓝牙设备
* 查询适配设备
* 建立RFCOMM通道
* 通过发现服务连接其他蓝牙设备
* 交换数据
* 管理多项连接

其中，本文只介绍简单的点对点蓝牙设备的数据交换技术，其余内容请自行查阅相关文档。此外本文只涉及经典蓝牙(Classic Bluetooth)的开发，并不涉及低功耗蓝牙(BLE)的开发。

**2.基础**

关于经典蓝牙（以下简称蓝牙）开发所用到的API都来自于android.bluetooth包中，本部分主要介绍相关类的概要说明。

**2.1BluetoothAdapter**

BluetoothAdapter类的对象代表本地的蓝牙适配器。BluetoothAdapter是所有蓝牙交互操作的入口点，通过使用该类的对象，可以完成以下操作：

* 发现其他蓝牙设备
* 查询已配对的设备
* 通过已知的MAC地址实例化远程蓝牙设备
* 创建BluetoothServerSocket类（下文2.4）对象监听与其他蓝牙设备的通信。

**2.2BluetoothDevice**

表示远程的蓝牙设备。使用该类对象可进行远程蓝牙设备的连接请求，以及查询该蓝牙设备的信息，例如名称，地址等。

**2.3BluetoothSocket**

表示蓝牙socket的接口（与TCP Socket类似， 关于socket的概念请自行查阅计算机网络的相关内容）。该类的对象作为应用中数据传输的连接点。

**2.4BluetoothServerSocket**

表示服务器socket，用来监听未来的请求（和TCP ServerSocket类似）。为了能使两个蓝牙设备进行连接，一个设备必须使用该类开启服务器socket，当远程的蓝牙设备请求该服务端设备时，如果连接被接受，BluetoothServerSocket将会返回一个已连接的BluetoothSocket类对象。

**2.5BluetoothClass**

描述蓝牙设备的主要特征。BluetoothClass的类对象是一个只读的蓝牙设备的属性集。尽管该类对象并不能可靠地描述BluetoothProfile的所有内容以及该设备支持的所有服务信息，但是该类对象仍然有助于对该设备的类型进行提示。

**2.6BluetoothProfile**

表示蓝牙规范，蓝牙规范是两个基于蓝牙设备通信的标准。

对于基本的蓝牙通信技术，上述类已经足够，更多内容请参照文档。

**3.权限**

**3.1android.permission.BLUETOOTH**

为了能够在你开发的应用设备中使用蓝牙功能，必须声明蓝牙的权限"BLUETOOTH"。在进行蓝牙的通信，例如请求连接，接受连接以及交换数据中，需要用到该权限[[1]](" \l "fn1)。

<manifest ... >

<uses-permission android:name="android.permission.BLUETOOTH" />

...

</manifest>

**3.2android.permission.BLUETOOTH\_ADMIN**

如果你的应用程序需要实例化蓝牙设备的搜索或者对蓝牙的设置进行操作，那么必须声明BLUETOOTH\_ADMIN权限。大多数应用需要该权限对本地的蓝牙设备进行搜索。该权限的其他能力并不应当被使用，除非你的应用是一个电源管理的应用，需要对蓝牙的设置进行修改[[2]](" \l "fn2)。

<manifest ... >

<uses-permission android:name="android.permission.BLUETOOTH\_ADMIN" />

...

</manifest>

**4.开启蓝牙(Setting Up Bluetooth)**

在你的应用程序能够使用蓝牙进行通信之前，你需要进行确认蓝牙设备是否被当前设备所支持。如果当前设备支持蓝牙，则需要请求开启蓝牙设备。该部分可使用BluetoothAdapter通过两步完成。

**4.1获得BluetoothAdapter**

BluetoothAdapter对于任何的蓝牙行为都是必备的。获取BluetoothAdapter，可通过调用静态方法getDefaultAdater()。该方法返回一个BluetoothAdapter对象，用来代表自己设备的蓝牙适配器。如果getDefaultAdapter()返回null，那么该设备并不支持蓝牙，而你对于本文的阅读也就到此为止了，请点击窗口右上角的关闭按钮╮(╯\_╰)╭。代码示例如下：

BluetoothAdapter mBluetoothAdapter = BluetoothAdapter.getDefaultAdapter();

if (mBluetoothAdapter == null) {

// Device does not support Bluetooth

}

**4.2开启蓝牙**

接下来，你需要确保蓝牙处于开启的状态。调用isEnabled()方法来检查蓝牙目前是否可用。如果该方法返回false，那么蓝牙处于不可用的状态。为了请求蓝牙设备的开启，使用ACTION\_REQUEST\_ENABLE的Intent，并调用startActivityForResult()方法。这将会通过系统设置开启你的蓝牙，例如：

if (!mBluetoothAdapter.isEnabled()) {

Intent enableBtIntent = new Intent(BluetoothAdapter.ACTION\_REQUEST\_ENABLE);

startActivityForResult(enableBtIntent, REQUEST\_ENABLE\_BT);

}

调用该方法后，一个请求开启蓝牙的对话框将会出现在屏幕上。如果用户点击确认，那么系统将会开启蓝牙设备，该过程完成（或失败）后，将会回到你的应用程序。

常量REQUEST\_ENABLE\_BT是本地定义的整型（需要大于0），当系统通过onActivityResult() 返回至你的应用程序时，将作为requestCode的参数。

如果成功开启了蓝牙，你的Activity将收到RESULT\_OK作为resultCode。如果蓝牙不能成功开启（例如用户选择“取消”），则resultCode为RESULT\_CANCELED[[3]](#fn3)。

可选择的是，你的应用也可以监听"ACTION\_STATE\_CHANGED"的广播Intent(不再赘述广播机制。新手请查阅文档或留言)。当蓝牙的状态发生变化时，系统将会进行广播。该广播包含两个额外的域，分别是：EXTRA\_STATE和EXTRA\_PREVIOUS\_STATE，分别包含蓝牙的新旧状态。可能的值为：STATE\_TURNING\_ON（开启中），STATE\_ON（已开启）， STATE\_TURNING\_OFF（关闭中）以及 STATE\_OFF（已关闭）。

开启扫描功能将会自动开启蓝牙设备。如果你始终想扫描，那么可以跳过4.2环节。关于如何开启扫描功能，详见后文。

**5.查找设备**

使用BluetoothAdapter，通过搜索设备或查询配对设备列表可以找到远程蓝牙设备。

设备搜索(Device Discovery)是一个扫描的过程，用来搜索本地开启蓝牙的设备，在此之后请求每一个扫描到设备的信息。然而，一个蓝牙设备只有处于可见状态下才会返回设备信息，例如设备名称，MAC地址等。使用该信息，设备能够实例化和该设备的蓝牙连接。

当第一次和远程蓝牙设备进行连接时，一个配对的请求将会自动呈现在用户面前。当设备配对时，设备的基础信息将会被保存并能够使用蓝牙的API进行读取。使用远程蓝牙设备的MAC地址，介于蓝牙设备间的连接将能够在任意时刻实例化，而不需要进行搜索操作（假定设备在蓝牙的通信范围内）[[4]](#fn4)。

请记住配对(paired)和连接(connected)是不同的。配对意味着两个蓝牙设备知道彼此的存在，并有一个共享的key用于验证，并且能够彼此建立加密的链接。连接则意味着设备当前共享同一个RFCOMM通道，并能够彼此交换数据。当前的蓝牙API要求在建立RFCOMM通道之前进行配对。

下文将会描述如何找到已配对的设备，以及如何发现新的设备。

Android设备默认情况下蓝牙是不可见的。用户可以使蓝牙在有限的时间内可见，或者应用可以在用户界面内请求用户开启可见性。具体步骤请参考后文。

**5.1查询配对设备**

在搜索设备之前，有必要查询已配对的设备集，来得知想要连接的设备是否已经配对。为了执行上述操作，可以调用getBondedDevices()方法。该方法返回一个BluetoothDevice的集合来代表配对设备。例如，你可以查询所有的配对设备并使用ArrarAdapter显示它们：

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());

}

}

需要使用BluetoothDevice进行连接实例化的唯一值是MAC地址。在上述的例子中，该部分作为ArrayAdapter的一部分呈现给用户。关于如何使用MAC地址实例化连接，请参考后文。

**5.2发现设备**

执行发现设备的操作，仅仅需要执行startDiscovery()方法。该过程是异步的，该方法将会立刻返回一个布尔值表明搜索是否已经开始。通常情况下，该搜索的过程调用12秒钟的查询，随后返回找到的设备。

你的应用程序必须使用ACTION\_FOUNDd的Intent注册一个BroadastReceiver。该Intent用来接受每一个查找到设备的信息。对于每一个设备，系统将会广播ACTION\_FOUND。该Intent包含两个额外域，EXTRA\_DEVICE 和 EXTRA\_CLASS。分别包含一个BluetoothDevice类对象和BluetoothClass类对象。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：**千万不要忘了在onDestroy()方法中进行反注销**

**执行设备搜索的操作是一项很繁重的任务，会消耗大量的资源。一旦你找到了一个设备并要进行连接，请务必确认是否停止搜索设备的操作。如果已经进行了连接，那么搜索操作将会显著地降低连接的速率，因此你应当在连接时停止搜索。可通过cancelDiscovery()方法停止搜索。**

**5.2.1开启搜索**

如果想要使自己的蓝牙设备可见，使用ACTION\_REQUEST\_DISCOVERABLE的Intent，调用startActivityForResult(Intent, int)方法即可。这将会通过系统设置请求开启搜索模式。默认情况下，设备将在120秒内可见。你可以定义不同的时间长度，通过添加Intent的extra： EXTRA\_DISCOVERABLE\_DURATION即可。该时长最大为3600秒，最小为0，超出该范围的值都会被设为120秒。其中，0表示设备始终处于可见状态。例如：

Intent discoverableIntent = new

Intent(BluetoothAdapter.ACTION\_REQUEST\_DISCOVERABLE);

discoverableIntent.putExtra(BluetoothAdapter.EXTRA\_DISCOVERABLE\_DURATION, 300);

startActivity(discoverableIntent);

一个请求蓝牙可见的对话框将会被显示出来。如果用户选择“是”，那么该设备将会在指定时间内可见，你的Activity将会在onActiviyResult()中返回和时限相同的result code。如果用户选择“否”，那么result code将为ESULT\_CANCELED。

如果蓝牙设备没有开启，在执行搜索操作时将会自动开启蓝牙设备。

设备将在指定时间内保持可见。如果你想要检查状态的变化，可以通过使用ACTION\_SCAN\_MODE\_CHANGED的Intent注册广播进行监听。该广播onReceive()的Intent包含两个额外域：EXTRA\_SCAN\_MODE 和 EXTRA\_PREVIOUS\_SCAN\_MODE，分别表示新旧状态。可能的值有：SCAN\_MODE\_CONNECTABLE\_DISCOVERABLE（可连接可见），SCAN\_MODE\_CONNECTABLE（可连接但不可见） 或 SCAN\_MODE\_NONE（不可连接不可见）。

如果仅仅是连接到远程蓝牙设备的话，你并不需要开启可见性。开启可见性仅仅在你的应用中作为服务端时才是必要的。因为其他蓝牙设备必须找到你的设备之后才能建立连接。

**6.连接设备**

为了在你的应用中让双方设备建立连接，你必须同时实现服务器端和客户端的机制。因为其中一个设备一定会开启服务器Socket，而另一个进行连接（使用作为服务器端的MAC地址进行连接）。当客户端和服务器端彼此拥有一个在同一个RFCOMM通道已连接的BluetoothSocket时便可以进行数据的交换。在每一端，都可以获得输入和输出流，从而可以开始数据的传输。该部分将在后文描述，本部分只描述如何初始化设备间的连接。

服务器端和客户端通过不同的方式获得BluetoothSocket。当一个连接接受(accept)的时候服务器端接收BluetoothSocket。而客户端则通过打开服务器端的RFCOMM通道得到BluetoothSocket。

一种实现技术是，应用程序同时实现客户端和服务器端。因此，每一个服务器端的程序拥有一个server socket并监听连接。当然，也可以在一个应用中实现服务器端的功能，而另一个应用中实现客户端的部分。

如果两个设备之前并没有配对过，那么Android的框架将会自动进行配对的请求通知。因此当尝试进行连接时，你的应用并不需要关心两台设备是否已经配对。你的RFCOMM连接将会被阻塞，直到用户成功配对，或因为用户拒绝配对而取消，或者配对失败以及超时等。

**6.1服务器端**

两个设备进行连接时，必须有一个设备通过BluetoothServerSocket作为服务器。该server socket的目的是监听未来的连接请求，并当该请求被接受时，提供一个已经连接的BluetoothSocket。当从BluetoothServerSocket获得BluetoothSocket时，BluetoothServerSocket**必须**被抛弃，除非你想要连接多个设备。

步骤如下：

1. 通过调用listenUsingRfcommWithServiceRecord(String, UUID)获取BluetoothServerSocket。

String是你服务的可辨别名称。系统将会自动写入一个新的“服务发现协议(Service Discovery Protocol 简称SDP)”数据库入口至你的设备，该名称可随意命名，通常情况下是应用名称。UUID也包括SDP的入口，并作为和客户端连接基础。当客户端尝试连接至服务端设备时，将会带有想要连接设备的独一无二的UUID。这些UUID必须匹配，目的是为了能够使连接被接受。

可通过网络上的诸多UUID生成器来获得UUID的字符串，然后通过fromString(String)方法获得。

1. 通过调用accept()，开始监听连接请求。

这是一个阻塞的调用，将会在抛出异常或者连接被接受时返回。连接只有在远程设备发送一个带有和服务器端已注册的UUID相匹配的连接请求时才会被接受。当连接成功时,accept()将会返回一个已经连接的BluetoothSocket。

1. 除非你想要接受多个连接，否则的话，调用close()进行关闭。

这将会释放server socket以及相关的资源，但是并不会关闭从accept()中返回的，已经连接的BluetoothSocket。和TCP/IP不同，RFCOMM仅仅允许在一个通道中同时存在一个客户端。因此大多数情况下，在获得BluetoothSocket后立即调用close()是很有必要的。

accept()方法不应当在主线程（UI线程）中执行，因为这是一个阻塞的调用，能够租住任何和程序的交互。通常情况下和BluetoothServerSocket以及BluetoothSocket有关的任何操作都应该在新的线程中进行。在另外的线程中调用close()方法将会撤销该阻塞方法调用并立即返回。请注意，BluetoothServerSocket和BluetoothSokcet中的任何方法都是线程安全的。

例子如下：

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 = mBluetoothAdapter.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) { }

}

}

在例子中，一旦连接被接受并获得BluetoothSocket后，应用立即将该BluetoothSocket发送至独立的线程并关闭BluetoothSocket，挑出循环。

注意到，当accept()返回BluetoothSocket时，socket已经连接了，因此不应该调用connect方法。

manageConnectedSocket()是一个虚构的方法，用来初始化数据传输的线程，将在后文介绍数据传输的部分。

一旦监听到连接并获得BluetoothSocket时，应当立即调用close()关闭BluetoothServerSocket。cancel()则为此提供了一个公共的方法。

**6.2客户端**

为了和服务器端连接，首先需要拥有一个代表远程服务器的BluetoothDevice对象。之后必须使用BluetoothDevice获得一个BluetoothSocket并初始化连接。

基本步骤如下。

1. 使用BluetoothDevice，通过调用createRfcommSocketToServiceRecord(UUID)得到BluetoothSocket。
2. 通过调用connect()方法初始化连接。

系统将会在远程服务器上查询匹配UUID的SDP。如果查询成功，将会共享RFCOMM通道用于连接，connect()方法将会返回。该方法是一个阻塞的调用。如果12秒钟内未能成功连接，该方法将会跑出一个异常。

因为connect()是一个阻塞的调用，因此该连接的过程总是应当在一个独立的线程中进行。

**你应当确保在调用connect()时设备没有执行搜索设备的操作。如果搜索设备也在同时进行，那么将会显著地降低连接速率，并很大程度上会连接失败。**

例子如下：

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

mBluetoothAdapter.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) { }

}

}

cancelDiscovery() 在连接建立之前被调用。你应当总是在连接前这么做。这样做是安全的，虽然没有检查是否在搜索设备。（如果想要进行检查，可以调用isDiscovering()）。

manageConnectedSocket()是一个虚构的方法，用来初始化数据传输的线程，将在后文介绍数据传输的部分。

在完成BluetoothSocket的处理后，始终记得调用close()方法来进行清理。

7.管理连接

当成功进行设备间的连接时，每一个设备都持有一个已连接的BluetoothSocket。这时终于可以进行数据的传输了。使用BluetoothSocket，数据的传输非常简单。

步骤如下：

1. 通过getInputStream()以及getOutputStream()分别获得输入输出流。
2. 通过read(byte[]) 和 write(byte[]) 读写数据。

仅此而已。

当然，还有一些细节需要考虑。首先需要一个专门的线程进行读写的操作。这是很重要的一点，因为read(byte[]) 和 write(byte[])方法都是阻塞调用的。read(byte[])将会阻塞，直到从流中读到数据。write(byte[])并不会经常阻塞，但如果远程设备没有足够快的调用读操作以及缓存已满时而被阻塞。你应当在该独立线程的主循环中进行数据的读取，并在该线程中一个独立的公有方法进行写的操作。

例子如下

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) { }

}

}

构造函数获得必要的流，一旦执行，线程将会等待数据从输入流中流出。当read(byte[])返回字节时，数据将通过父类的Handler被发送至Activity。之后再返回并等待更多的字节流。

发送数据则仅仅需要简单地调用线程的write()方法即可。

线程中的cancel（）方法很重要，因为连接可以随时在任意时间通过BluetoothSocket终止。该方法在结束使用蓝牙连接后，应当总是被调用。