蓝牙

本文内容

关键类

- `BluetoothAdapter`
- `BluetoothDevice`
- `BluetoothSocket`
- `BluetoothServerSocket`

相关示例

- 蓝牙聊天
- 蓝牙 HDP（健康设备配置文件）

Android 平台包含蓝牙网络堆栈支持，凭借此项支持，设备能以无线方式与其他蓝牙设备交换数据。应用框架提供了通过 Android Bluetooth API 访问蓝牙功能的途径。 这些 API 允许应用以无线方式连接到其他蓝牙设备，从而实现点到点和多点无线功能。

使用 Bluetooth API，Android 应用可执行以下操作：

- 扫描其他蓝牙设备
- 查询本地蓝牙适配器的配对蓝牙设备
- 建立 RFCOMM 通道
- 通过服务发现连接到其他设备
- 与其他设备进行双向数据传输
- 管理多个连接

本文将介绍如何使用*传统蓝牙*。传统蓝牙适用于电池使用强度较大的操作，例如 Android 设备之间的流式传输和通信等。 针对具有低功耗要求的蓝牙设备，Android 4.3（API 级别 18）中引入了面向低功耗蓝牙的 API 支持。 如需了解更多信息，请参阅低功耗蓝牙。

基础知识

本文将介绍如何使用 Android Bluetooth API 来完成使用蓝牙进行通信的四项主要任务：设置蓝牙、查找局部区域内的配对设备或可用设备、连接设备，以及在设备之间传输数据。

`android.bluetooth` 包中提供了所有 Bluetooth API。 下面概要列出了创建蓝牙连接所需的类和接口：

`BluetoothAdapter`

表示本地蓝牙适配器（蓝牙无线装置）。 `BluetoothAdapter` 是所有蓝牙交互的入口点。 利用它可以发现其他蓝牙设备，查询绑定（配对）设备的列表，使用已知的 MAC 地址实例化 `BluetoothDevice`，以及创建 `BluetoothServerSocket` 以侦听来自其他设备的通信。

`BluetoothDevice`

表示远程蓝牙设备。利用它可以通过 `BluetoothSocket` 请求与某个远程设备建立连接，或查询有关该设备的信息，例如设备的名称、地址、类和绑定状态等。

`BluetoothSocket`

表示蓝牙套接字接口（与 TCP `Socket` 相似）。这是允许应用通过 InputStream 和 OutputStream 与其他蓝牙设备交换数据的连接点。

`BluetoothServerSocket`

表示用于侦听传入请求的开放服务器套接字（类似于 TCP `ServerSocket`）。 要连接两台 Android 设备，其中一台设备必须使用此类开放一个服务器套接字。 当一台远程蓝牙设备向此设备发出连接请求时， `BluetoothServerSocket` 将会在接受连接后返回已连接的 `BluetoothSocket`。

`BluetoothClass`

描述蓝牙设备的一般特征和功能。 这是一组只读属性，用于定义设备的主要和次要设备类及其服务。 不过，它不能可靠地描述设备支持的所有蓝牙配置文件和服务，而是适合作为设备类型提示。

`BluetoothProfile`

表示蓝牙配置文件的接口。 *蓝牙配置文件*是适用于设备间蓝牙通信的无线接口规范。 免提配置文件便是一个示例。 如需了解有关配置文件的详细讨论，请参阅使用配置文件

`BluetoothHeadset`

提供蓝牙耳机支持，以便与手机配合使用。 其中包括蓝牙耳机和免提（1.5 版）配置文件。

`BluetoothA2dp`

定义高质量音频如何通过蓝牙连接和流式传输，从一台设备传输到另一台设备。“A2DP”代表高级音频分发配置文件。

`BluetoothHealth`

表示用于控制蓝牙服务的健康设备配置文件代理。

`BluetoothHealthCallback`

用于实现 `BluetoothHealth` 回调的抽象类。您必须扩展此类并实现回调方法，以接收关于应用注册状态和蓝牙通道状态变化的更新内容。

`BluetoothHealthAppConfiguration`

表示第三方蓝牙健康应用注册的应用配置，以便与远程蓝牙健康设备通信。

`BluetoothProfile.ServiceListener`

在 `BluetoothProfile` IPC 客户端连接到服务（即，运行特定配置文件的内部服务）或断开服务连接时向其发送通知的接口。

蓝牙权限

要在应用中使用蓝牙功能，必须声明蓝牙权限 `BLUETOOTH`。您需要此权限才能执行任何蓝牙通信，例如请求连接、接受连接和传输数据等。

如果您希望您的应用启动设备发现或操作蓝牙设置，则还必须声明 `BLUETOOTH_ADMIN` 权限。 大多数应用需要此权限仅仅为了能够发现本地蓝牙设备。 除非该应用是将要应用户请求修改蓝牙设置的“超级管理员”，否则不应使用此权限所授予的其他能力。 **注**：如果要使用 `BLUETOOTH_ADMIN` 权限，则还必须拥有 `BLUETOOTH` 权限。

在您的应用清单文件中声明蓝牙权限。例如：

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

如需了解有关声明应用权限的更多信息，请参阅 <uses-permission> 参考资料。

设置蓝牙

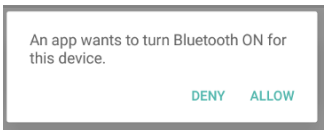

图 1：启用蓝牙对话框。

您需要验证设备支持蓝牙，并且如果支持确保将其启用，然后您的应用才能通过蓝牙进行通信。

如果不支持蓝牙，则应正常停用任何蓝牙功能。 如果支持蓝牙但已停用此功能，则可以请求用户在不离开应用的同时启用蓝牙。 可使用 `BluetoothAdapter`，分两步完成此设置：

1. 获取 `BluetoothAdapter`

   所有蓝牙 Activity 都需要 `BluetoothAdapter`。 要获取 `BluetoothAdapter`，请调用静态 `getDefaultAdapter()` 方法。这将返回一个表示设备自身的蓝牙适配器（蓝牙无线装置）的 `BluetoothAdapter`。 整个系统有一个蓝牙适配器，并且您的应用可使用此对象与之交互。 如果 `getDefaultAdapter()` 返回 null，则该设备不支持蓝牙，您的操作到此为止。 例如：

   ```
   BluetoothAdapter mBluetoothAdapter = BluetoothAdapter.getDefaultAdapter();
   if (mBluetoothAdapter == null) {
       // Device does not support Bluetooth
   }
   ```

2. 启用蓝牙

   下一步，您需要确保已启用蓝牙。调用 `isEnabled()` 以检查当前是否已启用蓝牙。 如果此方法返回 false，则表示蓝牙处于停用状态。要请求启用蓝牙，请使用 `ACTION_REQUEST_ENABLE` 操作 Intent 调用 `startActivityForResult()`。这将通过系统设置发出启用蓝牙的请求（无需停止您的应用）。 例如：

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

   如图 1 所示，将显示对话框，请求用户允许启用蓝牙。如果用户响应“Yes”，系统将开始启用蓝牙，并在该进程完成（或失败）后将焦点返回到您的应用。

   传递给 `startActivityForResult()` 的 `REQUEST_ENABLE_BT` 常量是在局部定义的整型（必须大于 0），系统会将其作为 `requestCode` 参数传递回您的 `onActivityResult()` 实现。

   如果成功启用蓝牙，您的 Activity 将会在 `onActivityResult()` 回调中收到 `RESULT_OK` 结果代码。 如果由于某个错误（或用户响应“No”）而没有启用蓝牙，则结果代码为 `RESULT_CANCELED`。

您的应用还可以选择侦听 `ACTION_STATE_CHANGED` 广播 Intent，每当蓝牙状态发生变化时，系统都会广播此 Intent。 此广播包含额外字段

EXTRA_STATE 和 EXTRA_PREVIOUS_STATE，二者分别包含新的和旧的蓝牙状态。 这些额外字段可能的值包括 STATE_TURNING_ON、STATE_ON、STATE_TURNING_OFF 和 STATE_OFF。侦听此广播适用于检测在您的应用运行期间对蓝牙状态所做的更改。

> **提示**：启用可检测性将会自动启用蓝牙。 如果您计划在执行蓝牙 Activity 之前一直启用设备的可检测性，则可以跳过上述步骤 2。 阅读下面的启用可检测性。

查找设备

使用 BluetoothAdapter，您可以通过设备发现或通过查询配对（绑定）设备的列表来查找远程蓝牙设备。

设备发现是一个扫描过程，它会搜索局部区域内已启用蓝牙功能的设备，然后请求一些关于各台设备的信息（有时也被称为“发现”、“查询”或“扫描”）。但局部区域内的蓝牙设备仅在其当前已启用可检测性时才会响应发现请求。 如果设备可检测到，它将通过共享一些信息（例如设备名称、类及其唯一 MAC 地址）来响应发现请求。 利用此信息，执行发现的设备可以选择发起到被发现设备的连接。

在首次与远程设备建立连接后，将会自动向用户显示配对请求。 设备完成配对后，将会保存关于该设备的基本信息（例如设备名称、类和 MAC 地址），并且可使用 Bluetooth API 读取这些信息。 利用远程设备的已知 MAC 地址可随时向其发起连接，而无需执行发现操作（假定该设备处于有效范围内）。

请记住，被配对与被连接之间存在差别。被配对意味着两台设备知晓彼此的存在，具有可用于身份验证的共享链路密钥，并且能够与彼此建立加密连接。 被连接意味着设备当前共享一个 RFCOMM 通道，并且能够向彼此传输数据。 当前的 Android Bluetooth API 要求对设备进行配对，然后才能建立 RFCOMM 连接。 （在使用 Bluetooth API 发起加密连接时，会自动执行配对）。

以下几节介绍如何查找已配对的设备，或使用设备发现来发现新设备。

> **注**：Android 设备默认处于不可检测到状态。 用户可通过系统设置将设备设为在有限的时间内处于可检测到状态，或者，应用可请求用户在不离开应用的同时启用可检测性。 下面讨论如何启用可检测性。

查询配对的设备

在执行设备发现之前，有必要查询已配对的设备集，以了解所需的设备是否处于已知状态。 为此，请调用 getBondedDevices()。 这将返回表示已配对设备的一组 BluetoothDevice。 例如，您可以查询所有已配对设备，然后使用 ArrayAdapter 向用户显示每台设备的名称：

```
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());
    }
}
```

要发起连接，BluetoothDevice 对象仅仅需要提供 MAC 地址。 在此示例中，它将保存为显示给用户的 ArrayAdapter 的一部分。 之后可提取该 MAC 地址，以便发起连接。 您可以在有关连接设备的部分详细了解如何创建连接。

发现设备

要开始发现设备，只需调用 startDiscovery()。该进程为异步进程，并且该方法会立即返回一个布尔值，指示是否已成功启动发现操作。 发现进程通常包含约 12 秒钟的查询扫描，之后对每台发现的设备进行页面扫描，以检索其蓝牙名称。

您的应用必须针对 ACTION_FOUND Intent 注册一个 BroadcastReceiver，以便接收每台发现的设备的相关信息。 针对每台设备，系统将会广播 ACTION_FOUND Intent。此 Intent 将携带额外字段 EXTRA_DEVICE 和 EXTRA_CLASS，二者分别包含 BluetoothDevice 和 BluetoothClass。例如，下面说明了在发现设备时如何注册以处理广播。

```
// 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
```

要发起连接，`BluetoothDevice` 对象仅仅需要提供 MAC 地址。 在此示例中，它将保存为显示给用户的 ArrayAdapter 的一部分。 之后可提取该 MAC 地址，以便发起连接。 您可以在有关连接设备的部分详细了解如何创建连接。

> **注意**：执行设备发现对于蓝牙适配器而言是一个非常繁重的操作过程，并且会消耗大量资源。 在找到要连接的设备后，确保始终使用 `cancelDiscovery()` 停止发现，然后再尝试连接。 此外，如果您已经保持与某台设备的连接，那么执行发现操作可能会大幅减少可用于该连接的带宽，因此不应该在处于连接状态时执行发现操作。

启用可检测性

如果您希望将本地设备设为可被其他设备检测到，请使用 `ACTION_REQUEST_DISCOVERABLE` 操作 Intent 调用 `startActivityForResult(Intent, int)`。 这将通过系统设置发出启用可检测到模式的请求（无需停止您的应用）。 默认情况下，设备将变为可检测到并持续 120 秒钟。 您可以通过添加 `EXTRA_DISCOVERABLE_DURATION` Intent Extra 来定义不同的持续时间。 应用可以设置的最大持续时间为 3600 秒，值为 0 则表示设备始终可检测到。 任何小于 0 或大于 3600 的值都会自动设为 120 秒。 例如，以下片段会将持续时间设为 300 秒：

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

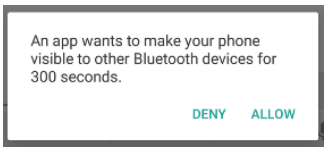

图 2：启用可检测性对话框。

如图 2 所示，将显示对话框，请求用户允许将设备设为可检测到。如果用户响应“Yes”，则设备将变为可检测到并持续指定的时间量。 然后，您的 Activity 将会收到对 `onActivityResult()` 回调的调用，其结果代码等于设备可检测到的持续时间。 如果用户响应“No”或出现错误，结果代码将为 `RESULT_CANCELED`。

> **注**：如果尚未在设备上启用蓝牙，则启用设备可检测性将会自动启用蓝牙。

设备将在分配的时间内以静默方式保持可检测到模式。如果您希望在可检测到模式发生变化时收到通知，您可以针对 `ACTION_SCAN_MODE_CHANGED` Intent 注册 BroadcastReceiver。 它将包含额外字段 `EXTRA_SCAN_MODE` 和 `EXTRA_PREVIOUS_SCAN_MODE`，二者分别告知您新的和旧的扫描模式。 每个字段可能的值包括 `SCAN_MODE_CONNECTABLE_DISCOVERABLE`、 `SCAN_MODE_CONNECTABLE` 或 `SCAN_MODE_NONE`，这些值分别指示设备处于可检测到模式、未处于可检测到模式但仍能接收连接，或未处于可检测到模式并且无法接收连接。

如果您将要发起到远程设备的连接，则无需启用设备可检测性。 仅当您希望您的应用托管将用于接受传入连接的服务器套接字时，才有必要启用可检测性，因为远程设备必须能够发现该设备，然后才能发起连接。

连接设备

要在两台设备上的应用之间创建连接，必须同时实现服务器端和客户端机制，因为其中一台设备必须开放服务器套接字，而另一台设备必须发起连接（使用服务器设备的 MAC 地址发起连接）。 当服务器和客户端在同一 RFCOMM 通道上分别拥有已连接的 `BluetoothSocket` 时，二者将被视为彼此连接。 这种情况下，每台设备都能获得输入和输出流式传输，并且可以开始传输数据，在有关管理连接的部分将会讨论这一主题。 本部分介绍如何在两台设备之间发起连接。

服务器设备和客户端设备分别以不同的方法获得需要的 `BluetoothSocket`。服务器将在传入连接被接受时收到套接字。 客户端将在其打开到服务器的 RFCOMM 通道时收到该套接字。

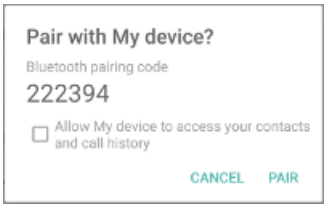

图 3：蓝牙配对对话框。

一种实现技术是自动将每台设备准备为一个服务器，从而使每台设备开放一个服务器套接字并侦听连接。然后任一设备可以发起与另一台设备的连接，并成为客户端。 或者，其中一台设备可显式“托管”连接并按需开放一个服务器套接字，而另一台设备则直接发起连接。

> **注**：如果两台设备之前尚未配对，则在连接过程中，Android 框架会自动向用户显示配对请求通知或对话框（如图 3 所示）。因此，在尝试连接设备时，您的应用无需担心设备是否已配对。 您的 RFCOMM 连接尝试将被阻塞，直至用户成功完成配对或配对失败（包括用户拒绝配对、配对失败或超时）。

连接为服务器

当您需要连接两台设备时，其中一台设备必须通过保持开放的 `BluetoothServerSocket` 来充当服务器。 服务器套接字的用途是侦听传入的连接请求，并在接受一个请求后提供已连接的 `BluetoothSocket`。 从 `BluetoothServerSocket` 获取 `BluetoothSocket` 后，可以（并且应该）舍弃 `BluetoothServerSocket`，除非您需要接受更多连接。

以下是设置服务器套接字并接受连接的基本过程：

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

   该字符串是您的服务的可识别名称，系统会自动将其写入到设备上的新服务发现协议 (SDP) 数据库条目（可使用任意名称，也可直接使用您的应用名称）。 UUID 也包含在 SDP 条目中，并且将作为与客户端设备的连接协议的基础。 也就是说，当客户端尝试连接此设备时，它会携带能够唯一标识其想要连接的服务的 UUID。 两个 UUID 必须匹配，在下一步中，连接才会被接受。

   > **关于 UUID**
   >
   > 通用唯一标识符 (UUID) 是用于唯一标识信息的字符串 ID 的 128 位标准化格式。 UUID 的特点是其足够庞大，因此您可以选择任意随机值而不会发生冲突。 在此示例中，它被用于唯一标识应用的蓝牙服务。 要获取 UUID 以用于您的应用，您可以使用网络上的众多随机 UUID 生成器之一，然后使用 `fromString(String)` 初始化一个 `UUID`。

2. 通过调用 `accept()` 开始侦听连接请求。

   这是一个阻塞调用。它将在连接被接受或发生异常时返回。 仅当远程设备发送的连接请求中所包含的 UUID 与向此侦听服务器套接字注册的 UUID 相匹配时，连接才会被接受。 操作成功后，`accept()` 将会返回已连接的 `BluetoothSocket`。

3. 除非您想要接受更多连接，否则请调用 `close()`。

   这将释放服务器套接字及其所有资源，但*不会*关闭 `accept()` 所返回的已连接的 `BluetoothSocket`。 与 TCP/IP 不同，RFCOMM 一次只允许每个通道有一个已连接的客户端，因此大多数情况下，在接受已连接的套接字后立即在 `BluetoothServerSocket` 上调用 `close()` 是行得通的。

`accept()` 调用不应在主 Activity UI 线程中执行，因为它是阻塞调用，并会阻止与应用的任何其他交互。 在您的应用所管理的新线程中使用 `BluetoothServerSocket` 或 `BluetoothSocket` 完成所有工作，这通常是一种行之有效的做法。 要终止 `accept()` 等被阻塞的调用，请通过另一个线程在 `BluetoothServerSocket`（或 `BluetoothSocket`）上调用 `close()`，被阻塞的调用将会立即返回。 请注意，`BluetoothServerSocket` 或 `BluetoothSocket` 中的所有方法都是线程安全的方法。

示例

以下是一个用于接受传入连接的服务器组件的简化线程：

```
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) { }
    }
}
```

在此示例中，只需要一个传入连接，因此在接受连接并获取 `BluetoothSocket` 之后，应用会立即将获取的 `BluetoothSocket` 发送到单独的线程，关闭 `BluetoothServerSocket` 并中断循环。

请注意，当 `accept()` 返回 `BluetoothSocket` 时，表示套接字已连接好，因此您*不应该*像在客户端那样调用 `connect()`。

`manageConnectedSocket()` 是应用中的虚构方法，它将启动用于传输数据的线程，在有关管理连接的部分将会讨论这一主题。

在完成传入连接的侦听后，通常应立即关闭您的 `BluetoothServerSocket`。 在此示例中，获取 `BluetoothSocket` 后立即调用 `close()`。您也可能希望在您的线程中提供一个公共方法，以便在需要停止侦听服务器套接字时关闭私有 `BluetoothSocket`。

连接为客户端

要发起与远程设备（保持开放的服务器套接字的设备）的连接，必须首先获取表示该远程设备的 `BluetoothDevice` 对象。（在前面有关查找设备的部分介绍了如何获取 `BluetoothDevice`）。 然后必须使用 `BluetoothDevice` 来获取 `BluetoothSocket` 并发起连接。

以下是基本过程：

1. 使用 `BluetoothDevice`，通过调用 `createRfcommSocketToServiceRecord(UUID)` 获取 `BluetoothSocket`。

   这将初始化将要连接到 `BluetoothDevice` 的 `BluetoothSocket`。 此处传递的 UUID 必须与服务器设备在使用 `listenUsingRfcommWithServiceRecord(String, UUID)` 开放其 `BluetoothServerSocket` 时所用的 UUID 相匹配。 要使用相同的 UUID，只需将该 UUID 字符串以硬编码方式编入应用，然后通过服务器代码和客户端代码引用该字符串。

2. 通过调用 `connect()` 发起连接。

   执行此调用时，系统将会在远程设备上执行 SDP 查找，以便匹配 UUID。 如果查找成功并且远程设备接受了该连接，它将共享 RFCOMM 通道以便在连接期间使用，并且 `connect()` 将会返回。 此方法为阻塞调用。 如果由于任何原因连接失败或 `connect()` 方法超时（大约 12

秒之后），它将会引发异常。

由于 `connect()` 为阻塞调用，因此该连接过程应始终在主 Activity 线程以外的线程中执行。

> 注：在调用 `connect()` 时，应始终确保设备未在执行设备发现。 如果正在进行发现操作，则会大幅降低连接尝试的速度，并增加连接失败的可能性。

示例

以下是发起蓝牙连接的线程的基本示例：

```
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) { }
    }
}
```

请注意，在建立连接之前会调用 `cancelDiscovery()`。 在进行连接之前应始终执行此调用，而且调用时无需实际检查其是否正在运行（但如果您确实想要执行检查，请调用 `isDiscovering()`）。

`manageConnectedSocket()` 是应用中的虚构方法，它将启动用于传输数据的线程，在有关管理连接的部分将会讨论这一主题。

在完成 `BluetoothSocket` 后，应始终调用 `close()` 以执行清理操作。这样做会立即关闭已连接的套接字并清理所有内部资源。

管理连接

在成功连接两台（或更多台）设备后，每台设备都会有一个已连接的 `BluetoothSocket`。 这一点非常有趣，因为这表示您可以在设备之间共享数据。 利用 `BluetoothSocket`，传输任意数据的一般过程非常简单：

1. 获取 `InputStream` 和 `OutputStream`，二者分别通过套接字以及 `getInputStream()` 和 `getOutputStream()` 来处理数据传输。

2. 使用 `read(byte[])` 和 `write(byte[])` 读取数据并写入到流式传输。

就这么简单。

当然，还需要考虑实现细节。首要的是，应该为所有流式传输读取和写入操作使用专门的线程。 这一点很重要，因为 `read(byte[])` 和 `write(byte[])` 方法都是阻塞调用。`read(byte[])` 将会阻塞，直至从流式传输中读取内容。`write(byte[])` 通常不会阻塞，但如果远程设备没有足够快地调用 `read(byte[])`，并且中间缓冲区已满，则其可能会保持阻塞状态以实现流量控制。因此，线程中的主循环应专门用于读取 `InputStream`。 可使用线程中单独的公共方法来发起对 `OutputStream` 的写入操作。

示例

以下是可能的显示内容示例：

```
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) { }
    }
}
```

构造函数获取必要的流式传输，并且一旦执行，线程将会等待通过 InputStream 传入的数据。 当 `read(byte[])` 返回流式传输中的字节时，将

使用来自父类的成员处理程序将数据发送到主 Activity。 然后该方法返回并等待流式传输提供更多字节。

发送传出数据不外乎从主 Activity 调用线程的 `write()` 方法，并传入要发送的字节。 然后，此方法直接调用 `write(byte[])`，将数据发送到远程设备。

线程的 `cancel()` 方法很重要，它能通过关闭 `BluetoothSocket` 随时终止连接。当您结束蓝牙连接的使用时，应始终调用此方法。

有关使用 Bluetooth API 的演示，请参阅蓝牙聊天示例应用。

使用配置文件

从 Android 3.0 开始，Bluetooth API 便支持使用蓝牙配置文件。 *蓝牙配置文件*是适用于设备间蓝牙通信的无线接口规范。 免提配置文件便是一个示例。 对于连接到无线耳机的手机，两台设备都必须支持免提配置文件。

您可以实现接口 `BluetoothProfile`，通过写入自己的类来支持特定的蓝牙配置文件。 Android Bluetooth API 提供了以下蓝牙配置文件的实现：

- **耳机**。耳机配置文件提供了蓝牙耳机支持，以便与手机配合使用。 Android 提供了 `BluetoothHeadset` 类，它是用于通过进程间通信 (IPC) 来控制蓝牙耳机服务的代理。 这包括蓝牙耳机和免提（1.5 版）配置文件。 `BluetoothHeadset` 类包含 AT 命令支持。 有关此主题的详细讨论，请参阅供应商特定的 AT 命令
- **A2DP**。高级音频分发配置文件 (A2DP) 定义了高质量音频如何通过蓝牙连接和流式传输，从一个设备传输到另一个设备。 Android 提供了 `BluetoothA2dp` 类，它是用于通过 IPC 来控制蓝牙 A2DP 服务的代理。
- **健康设备**。Android 4.0（API 级别 14）引入了对蓝牙健康设备配置文件 (HDP) 的支持。 这允许您创建应用，使用蓝牙与支持蓝牙功能的健康设备进行通信，例如心率监测仪、血糖仪、温度计、台秤等等。 有关支持的设备列表及其相应的设备数据专业化代码，请参阅 www.bluetooth.org 上的**蓝牙分配编号**。 请注意，这些值在 ISO/IEEE 11073-20601 [7] 规范的“命名法规附录”中也被称为 MDC_DEV_SPEC_PROFILE_*。 有关 HDP 的详细讨论，请参阅健康设备配置文件。

以下是使用配置文件的基本步骤：

1. 获取默认适配器（请参阅设置蓝牙）。
2. 使用 `getProfileProxy()` 建立到配置文件所关联的配置文件代理对象的连接。在以下示例中，配置文件代理对象是一个 `BluetoothHeadset` 的实例。
3. 设置 `BluetoothProfile.ServiceListener`。此侦听程序会在 `BluetoothProfile` IPC 客户端连接到服务或断开服务连接时向其发送通知。
4. 在 `onServiceConnected()` 中，获取配置文件代理对象的句柄。
5. 获得配置文件代理对象后，可以立即将其用于监视连接状态和执行其他与该配置文件相关的操作。

例如，以下代码片段显示了如何连接到 `BluetoothHeadset` 代理对象，以便能够控制耳机配置文件：

```
BluetoothHeadset mBluetoothHeadset;

// Get the default adapter
BluetoothAdapter mBluetoothAdapter = BluetoothAdapter.getDefaultAdapter();

// Establish connection to the proxy.
mBluetoothAdapter.getProfileProxy(context, mProfileListener, BluetoothProfile.HEADSET);

private BluetoothProfile.ServiceListener mProfileListener = new BluetoothProfile.ServiceListener() {
    public void onServiceConnected(int profile, BluetoothProfile proxy) {
        if (profile == BluetoothProfile.HEADSET) {
            mBluetoothHeadset = (BluetoothHeadset) proxy;
        }
    }
    public void onServiceDisconnected(int profile) {
        if (profile == BluetoothProfile.HEADSET) {
            mBluetoothHeadset = null;
        }
    }
};

// ... call functions on mBluetoothHeadset

// Close proxy connection after use.
mBluetoothAdapter.closeProfileProxy(mBluetoothHeadset);
```

供应商特定的 AT 命令

从 Android 3.0 开始，应用可以注册接收耳机所发送的预定义的供应商特定 AT 命令的系统广播（例如 Plantronics +XEVENT 命令）。 例如，应用可以接收指示所连接设备的电池电量的广播，并根据需要通知用户或采取其他操作。 为 `ACTION_VENDOR_SPECIFIC_HEADSET_EVENT` intent 创建广播接收器，以处理耳机的供应商特定 AT 命令。

健康设备配置文件

Android 4.0（API 级别 14）引入了对蓝牙健康设备配置文件 (HDP) 的支持。 这允许您创建应用，使用蓝牙与支持蓝牙功能的健康设备进行通信，例如心率监测仪、血糖仪、温度计、台秤等等。 Bluetooth Health API 包括类 `BluetoothHealth`、`BluetoothHealthCallback` 和 `BluetoothHealthAppConfiguration`，在基础知识部分介绍了这些类。

在使用 Bluetooth Health API 时，了解以下关键 HDP 概念很有帮助：

概念	说明
源设备	在 HDP 中定义的角色。*源设备*是将医疗数据传输到 Android 手机或平板电脑等智能设备的健康设备（体重秤、血糖仪、温度计等）。
汇集设备	在 HDP 中定义的角色。在 HDP 中，*汇集设备*是接收医疗数据的智能设备。 在 Android HDP 应用中，汇集设备表示为 `BluetoothHealthAppConfiguration` 对象。
注册	指的是注册特定健康设备的汇集设备。
连接	指的是开放健康设备与 Android 手机或平板电脑等智能设备之间的通道。

创建 HDP 应用

以下是创建 Android HDP 应用所涉及的基本步骤：

1. 获取 `BluetoothHealth` 代理对象的引用。

   与常规耳机和 A2DP 配置文件设备相似，您必须使用 `BluetoothProfile.ServiceListener` 和 `HEALTH` 配置文件类型来调用 `getProfileProxy()`，以便与配置文件代理对象建立连接。

2. 创建 `BluetoothHealthCallback` 并注册充当健康汇集设备的应用配置 (`BluetoothHealthAppConfiguration`)。

3. 建立到健康设备的连接。一些设备将会发起该连接。 对于这类设备，无需执行该步骤。

4. 成功连接到健康设备后，使用文件描述符对健康设备执行读/写操作。

接收的数据需要使用实现了 IEEE 11073-xxxxx 规范的健康管理器进行解释。

5. 完成后，关闭健康通道并取消注册该应用。该通道在长期闲置时也会关闭。

有关描述上述步骤的完整代码示例，请参阅蓝牙 HDP（健康设备配置文件）。