Bluetooth(3.0/4.0) Using

Android 手机 Bluetooth 编程

邹继强

2014.06.24

本文介绍蓝牙 3.0、2.0 及蓝牙 4.0 版本 Android 编程应用方法

目录

1 蓝牙简介

蓝牙这个名称来自于第十世纪的一位丹麦国王哈拉尔蓝牙王，Blatand 在英文里的意思

图 1 哈拉尔蓝牙王

可以被解释为 Bluetooth(蓝牙)因为国王喜欢吃蓝莓，牙龈每天都是蓝色的所以叫蓝牙。在行业协会筹备阶段，需要一个极具有表现力的名字来命名这项高新技术。行业组织人员，在经过一夜关于欧洲历史和未来无线技术发展的讨论后，有些人认为用 Blatand 国王的名字命名再合适不过了。Blatand 国王将挪威，瑞典和丹麦统一起来；他的口齿伶俐,善于交际,就如同这项即将面世的技术，技术将被定义为允许不同工业领域之间的协调工作，保持着各个系统领域之间的良好交流，例如计算机，手机和汽车行业之间的工作。名字于是就这么定下来了。

蓝牙的创始人是爱立信公司，爱立信早在 1994 年就已进行研发。1997 年，爱立信与其他设备生产商联系，并激发了他们对该项技术的浓厚兴趣。 1998 年 2 月，跨国大公司，

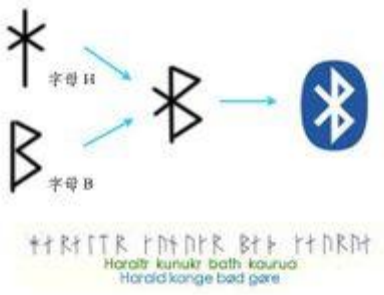


图 2 蓝牙标志的来历

包括诺基亚、苹果、三星组成的一个特殊兴趣小组（SIG），他们共同的目标是建立一个全球性的小范围无线通信技术，即蓝牙。

而蓝牙这个标志的设计：它取自 Harald Bluetooth 名字中的「H」和「B」两个字母，用古北欧字母来表示，将这两者结合起来，就成为了蓝牙的 logo（见图）。

2 蓝牙的工作原理

2.1 蓝牙通信的主从关系

蓝牙技术规定每一对设备之间进行蓝牙通讯时，必须一个为主角色，另一为从角色，才能进行通信，通信时，必须由主端进行查找，发起配对，建链成功后，双方即可收发数据。理论上，一个蓝牙主端设备，可同时与 7 个蓝牙从端设备进行通讯。一个具备蓝牙通讯功能的设备， 可以在两个角色间切换，平时工作在从模式，等待其它主设备来连接，需要时，转换为主模式，向其它设备发起呼叫。一个蓝牙设备以主模式发起呼叫时，需要知道对方的蓝牙地址，配对密码等信息，配对完成后，可直接发起呼叫。

2.2 蓝牙的呼叫过程

蓝牙主端设备发起呼叫，首先是查找，找出周围处于可被查找的蓝牙设备。主端设备找到从端蓝牙设备后，与从端蓝牙设备进行配对，此时需要输入从端设备的 PIN 码，也有设备不需要输入 PIN 码。配对完成后，从端蓝牙设备会记录主端设备的信任信息，此时主端即可向从端设备发起呼叫，已配对的设备在下次呼叫时，不再需要重新配对。已配对的设备，做为从端的蓝牙耳机也可以发起建链请求，但做数据通讯的蓝牙模块一般不发起呼叫。链路建立成功后，主从两端之间即可进行双向的数据或语音通讯。在通信状态下，主端和从端设备都可以发起断链，断开蓝牙链路。

2.3 蓝牙一对一的串口数据传输应用

蓝牙数据传输应用中，一对一串口数据通讯是最常见的应用之一，蓝牙设备在出厂前即提前设好两个蓝牙设备之间的配对信息，主端预存有从端设备的 PIN 码、地址等，两端设备加电即自动建链，透明串口传输，无需外围电路干预。一对一应用中从端设备可以设为两种类型，一是静默状态，即只能与指定的主端通信，不被别的蓝牙设备查找；二是开发状态，既可被指定主端查找，也可以被别的蓝牙设备查找建链。

3 蓝牙 Android 编程应用

参考文献：http://www.epx.com.br/artigos/bluetooth_gatt.php

3.1 蓝牙 3.0 及以下版本编程

3.1.1 认识一下 UUID

UUID 含义是通用唯一识别码 (Universally Unique Identifier)，这 是一个软件建构的标准，也是被开源软件基金会 (Open Software Foundation, OSF) 的组织应用在分布式计算

环境 (Distributed Computing Environment, DCE) 领域的一部分。

在蓝牙 3.0 及一下版本中，UUID 被用于唯一标识一个服务，比如文件传输服务，串口服务、打印机服务等，如下：

#蓝牙串口服务
SerialPortServiceClass_UUID = '{00001101-0000-1000-8000-00805F9B34FB}'
LANAccessUsingPPPServiceClass_UUID = '{00001102-0000-1000-8000-00805F9B34FB}'
#拨号网络服务
DialupNetworkingServiceClass_UUID = '{00001103-0000-1000-8000-00805F9B34FB}'
#信息同步服务
IrMCSyncServiceClass_UUID = '{00001104-0000-1000-8000-00805F9B34FB}'
SDP_OBEXObjectPushServiceClass_UUID = '{00001105-0000-1000-8000-00805F9B34FB}
'
#文件传输服务
OBEXFileTransferServiceClass_UUID = '{00001106-0000-1000-8000-00805F9B34FB}'
IrMCSyncCommandServiceClass_UUID = '{00001107-0000-1000-8000-00805F9B34FB}'

蓝牙的连接有主从设备，提供服务的可以认为是从设备。主设备通过 UUID 访问从设备提供具有相同 UUID 的服务，从而建立客服端—服务器（C/S）模式。

3.1.2 版本 3.0 蓝牙 Android 编程原理

使用蓝牙功能 ，首先要获得权限，蓝牙权限设置：

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

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

A 建立并获得本地蓝牙适配器：

```
mBluetoothAdapter = BluetoothAdapter.getDefaultAdapter();
```

B 如果本地有蓝牙设备，者开启。

```
if (mBluetoothAdapter != null) {   //判断是否有蓝牙

    if (!mBluetoothAdapter.isEnabled()) {//判断蓝牙 是否开启，未开启则请用户开启

Intent enableBtIntent = new
Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);

   startActivityForResult(enableBtIntent, 1);
```

```
}
```

C 搜索已配对设备，如果有则添加到配对列表中

```
Set<BluetoothDevice> pairedDevices =
          mBluetoothAdapter.getBondedDevices();
        // If there are paired devices
        if (pairedDevices.size() > 0) {
            // Loop through paired devices
            for (BluetoothDevice device : pairedDevices) {
        mArrayAdapter.add(device.getName() + "\n" + device.getAddress());

            }
        }
```

D 搜索未配对设备，添加到未配对列表

mBluetoothAdapter.startDiscovery();//开始收索

搜索接收函数：

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

收索接收函数需要注册：

```
// Register the BroadcastReceiver
IntentFilter filter = new IntentFilter(BluetoothDevice.ACTION_FOUND);
        registerReceiver(mReceiver, filter); // Don't forget to unregister
during onDestroy
```

F 如果是服务器端，需要建立监听，注意监听的是某个服务的ＵＵＩＤ，服务器监听类如下：

```
    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("blue",MY_UUID);
// MY_UUID= UUID.fromString("0000ffe1-0000-1000-8000-00805f9b34fb");
        } 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)
             ConnectedThread ced=new ConnectedThread(socket);//监听到则建立
连接
             ced.start();
                //manageConnectedSocket(socket);
             try
             {
                 mmServerSocket.close();
             }
             catch(Exception e)
             {}
                break;
            }
        }
    }
    /** Will cancel the listening socket, and cause the thread to finish */
    public void cancel() {
        try {
```

```
                mmServerSocket.close();
            } catch (IOException e) { }
        }
    }
```

G 如果是客户端，则需要直接通过UUID连接服务端，客户端连接类如下：

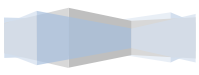
```
    private class ConnectThread extends Thread {
        private final BluetoothSocket mmSocket;
        private final BluetoothDevice mmDevice;
        public ConnectThread(BluetoothDevice device) {
            // Use a temporary object that is later assigned to mmSocket,
            // because 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();
                ConnectedThread ced=new ConnectedThread(mmSocket);
                ced.start();
            }
            catch (Exception e) {
             Log.e("connect0e",e.toString());
             //t1.append("\r\n 连接失败001" );
                // Unable to connect; close the socket and get out
                try {
                    mmSocket.close();
                }
                catch (Exception e1)
                {
                 Log.e("close",e1.toString());
                }
            }
            // 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 {
```

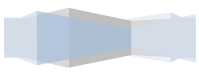
```
                mmSocket.close();
            } catch (IOException e) { }
        }
    }
```

H　客户端与服务器端建立连接成功后，需要ConnectedThread类接收发送数据：

```
    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();
                    String str= new String(buffer,"ISO-8859-1");
                    str=str.substring(0,bytes);
                    Log.e("read",str);
                } 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) { }
      }
    }
```

3.2 蓝牙 4.0 编程

3.2.1 概念

Generic Attribute Profile (GATT)

通过 BLE 连接，读写属性类小数据的 Profile 通用规范。现在所有的 BLE 应用 Profile 都是基于 GATT 的。蓝牙 4.0 特有。

Attribute Protocol (ATT)

GATT 是基于 ATT Protocol 的。ATT 针对 BLE 设备做了专门的优化，具体就是在传输过程中使用尽量少的数据。每个属性都有一个唯一的 UUID，属性将以 characteristics and services 的形式传输。

Characteristic

Characteristic 可以理解为一个数据类型，它包括一个 value 和 0 至多个对次 value 的描述（Descriptor）。

Descriptor

对 Characteristic 的描述，例如范围、计量单位等。

Service

Characteristic 的集合。例如一个 service 叫做“Heart Rate Monitor”，它可能包含多个 Characteristics，其中可能包含一个叫做“heart rate measurement"的 Characteristic。

3.2.2 编程 API –Level18-19

A 权限设置

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

除了蓝牙权限外，如果需要 BLE feature 则还需要声明 uses-feature：

```
<uses-feature android:name="android.hardware.bluetooth_le"
android:required="true"/>
```

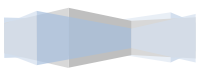
B 获得并开启本地蓝牙

```
final BluetoothManager bluetoothManager = (BluetoothManager)
getSystemService(Context.BLUETOOTH_SERVICE);

mBluetoothAdapter = bluetoothManager.getAdapter();

    if (mBluetoothAdapter == null || !mBluetoothAdapter.isEnabled()) {

    Intent enableBtIntent = new
    Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);

            startActivityForResult(enableBtIntent, 1);

    }
```

C 浏览提供服务的外设

```
private static final long SCAN_PERIOD = 10000;

private void scanLeDevice(final boolean enable) {

        if (enable) {

            // Stops scanning after a pre-defined scan period.

            mHandler.postDelayed(new Runnable() {

                @Override

                public void run() {

                    mScanning = false;
```

```
                    mBluetoothAdapter.stopLeScan(mLeScanCallback);
                }
            }, SCAN_PERIOD);
            mScanning = true;
            mBluetoothAdapter.startLeScan(mLeScanCallback);
        }
        else {
            mScanning = false;
            mBluetoothAdapter.stopLeScan(mLeScanCallback);
        }
    }
private BluetoothAdapter.LeScanCallback mLeScanCallback =
        new BluetoothAdapter.LeScanCallback() {
    @Override
    public void onLeScan(final BluetoothDevice device, int rssi,
            byte[] scanRecord) {
        runOnUiThread(new Runnable() {
           @Override
           public void run() {
            Log.e("find",device.getName());
//在此处可建立ArrayList<BluetoothDevice> mLeDevices保存外设,后续使用
            }
        });
    }
```

```
};
```

D 连接 GATT 外设并输入输出

以后所有的结果都会在这里得到，故特别关注！

- 连接 Gatt，此函数附带回调函数 ：mGattCallback(),若连接成功 ，则会触发回调函数 onConnectionStateChange();

```
mBluetoothGatt = device.connectGatt(context0, false, mGattCallback);
```

- 获得某个 Gatt 外设提供的服务

```
mBluetoothGatt.discoverServices();

onServicesDiscovered(BluetoothGatt gatt, int status);//回调
```

- 获得一个外设的所有服务，存在一个 arraylist<>中

```
BlueServiceList=gatt.getServices();
```

- 通过UUID获得某个服务

```
BluetoothGattService gattS =gattDevice.getService(UUID uuid);
```

- 通过UUID获得某个外设某个服务的一个Characteristic

```
characteristic = gattS.getCharacteristic(UUID uuid);
```

- 设置当characteristic改变时通知，用来从外设获得发来的值

```
gatt.setCharacteristicNotification(characteristic, true);
```

- 设备发来数据时回调函数

```
onCharacteristicChanged(BluetoothGatt gatt,

BluetoothGattCharacteristic characteristic)
```

- 发送数据到设备

```
characteristic.setValue("0123");

gatt.writeCharacteristic(characteristic);
```

```
//连接回调函数

    private final BluetoothGattCallback mGattCallback = new
```

```
BluetoothGattCallback() {

    @Override

    public void onConnectionStateChange(BluetoothGatt gatt, int status, int
newState) {

        String intentAction;

        if (newState == BluetoothProfile.STATE_CONNECTED) {

            intentAction = ACTION_GATT_CONNECTED;

            mConnectionState = STATE_CONNECTED;

           //broadcastUpdate(intentAction);//发送状态


            Log.d("连接", "Connected to GATT server.");

            // Attempts to discover services after successful connection.

            Log.d("", "Attempting to start service discovery:" +

                    mBluetoothGatt.discoverServices());//


        } else if (newState == BluetoothProfile.STATE_DISCONNECTED) {

            intentAction = ACTION_GATT_DISCONNECTED;

            mConnectionState = STATE_DISCONNECTED;

            Log.i("", "Disconnected from GATT server.");

           // broadcastUpdate(intentAction);

        }

    }

    @Override

    public void onServicesDiscovered(BluetoothGatt gatt, int status) {
```

```
        if (status == BluetoothGatt.GATT_SUCCESS) {
          //  broadcastUpdate(ACTION_GATT_SERVICES_DISCOVERED);
           Log.e("discovered1", "onServicesDiscovered received: " +
status);
            BlueServiceList=gatt.getServices();
           for(BluetoothGattService BS:BlueServiceList)
           {
                Log.e("服务",BS.getUuid().toString());

                if(n==0)
                {

                    gattDevice=gatt;

BluetoothGattService gattS
=gattDevice.getService(UUID.fromString("0000ffe0-0000-1000-8000-00805f9
b34fb"));
                    if(gattS!=null)
                    {
                        characteristic
=gattS.getCharacteristic(UUID.fromString("0000ffe1-0000-1000-8000-00805
f9b34fb"));
    gattDevice.setCharacteristicNotification(characteristic, true);
                        n++;
                    }
                }
```

```
                }
            }
            else
            {
            //Log.e("discovered0", "onServicesDiscovered received: " +
status);
            }
        }
        @Override
        public void onCharacteristicRead(BluetoothGatt gatt,
                                         BluetoothGattCharacteristic
characteristic,
                                         int status) {
            if (status == BluetoothGatt.GATT_SUCCESS) {
            //    broadcastUpdate(ACTION_DATA_AVAILABLE, characteristic);
            byte[] data = characteristic.getValue();
            Log.e("onCharacteristicRead", "onCharacteristicRead received: " +
new String(data));
            //gatt.readCharacteristic(characteristic);
            }
        }
        @Override
        public void onCharacteristicChanged(BluetoothGatt gatt,
                                            BluetoothGattCharacteristic
characteristic) {
```

```
            byte[] data = characteristic.getValue();

                Log.e("onCharacteristicChanged", "onCharacteristicChanged
received: " + new String(data));

        }

        @Override

        public void onCharacteristicWrite (BluetoothGatt gatt,
BluetoothGattCharacteristic characteristic, int status){

         Log.e("onCharacteristicWrite", "test----onCharacteristicWrite" );

        }

    };
```