**初识低功耗蓝牙**

Android 4.3（API Level 18）开始引入Bluetooth Low Energy（BLE，低功耗蓝牙）的核心功能并提供了相应的 API， 应用程序通过这些 API 扫描蓝牙设备、查询 services、读写设备的 characteristics（属性特征）等操作。

Android BLE 使用的蓝牙协议是 GATT 协议，有关该协议的详细内容可以参见[蓝牙官方文档](https://link.jianshu.com?t=https:/www.bluetooth.com/specifications/generic-attributes-overview" \t "_blank)。以下我引用一张官网的图来大概说明 Android 开发中我们需要了解的一些 Bluetooth Low Energy 的专业术语。

蓝牙协议图

**Service**

一个低功耗蓝牙设备可以定义许多 Service, Service 可以理解为一个功能的集合。设备中每一个不同的 Service 都有一个 128 bit 的 UUID 作为这个 Service 的独立标志。蓝牙核心规范制定了两种不同的UUID，一种是基本的UUID，一种是代替基本UUID的16位UUID。所有的蓝牙技术联盟定义UUID共用了一个基本的UUID：  
0x0000xxxx-0000-1000-8000-00805F9B34FB

为了进一步简化基本UUID，每一个蓝牙技术联盟定义的属性有一个唯一的16位UUID，以代替上面的基本UUID的‘x’部分。例如，心率测量特性使用0X2A37作为它的16位UUID，因此它完整的128位UUID为：

0x00002A37-0000-1000-8000-00805F9B34FB

**Characteristic**

在 Service 下面，又包括了许多的独立数据项，我们把这些独立的数据项称作 Characteristic。同样的，每一个 Characteristic 也有一个唯一的 UUID 作为标识符。在 Android 开发中，建立蓝牙连接后，我们说的通过蓝牙发送数据给外围设备就是往这些 Characteristic 中的 Value 字段写入数据；外围设备发送数据给手机就是监听这些 Charateristic 中的 Value 字段有没有变化，如果发生了变化，手机的 BLE API 就会收到一个监听的回调。

更详细的内容可以参见

[GATT Profile 简介](https://link.jianshu.com?t=https:/www.bluetooth.com/specifications/adopted-specifications)

[通用属性配置文件（GATT）及其服务，特性与属性介绍](https://link.jianshu.com?t=http:/blog.chinaunix.net/uid-21411227-id-5750680.html)

[GATT specification](https://link.jianshu.com?t=https:/www.bluetooth.com/specifications/generic-attributes-overview)

[GATT Services](https://link.jianshu.com?t=https:/www.bluetooth.com/specifications/gatt/services)

[蓝牙【GATT】协议介绍](https://link.jianshu.com?t=http:/blog.csdn.net/u013378580/article/details/52891462)

**Android BLE API 简介**

[BluetoothAdapter](https://link.jianshu.com?t=https:/developer.android.com/reference/android/bluetooth/BluetoothAdapter.html)

BluetoothAdapter 拥有基本的蓝牙操作，例如开启蓝牙扫描，使用已知的 MAC 地址 （BluetoothAdapter#getRemoteDevice）实例化一个 BluetoothDevice 用于连接蓝牙设备的操作等等。

[BluetoothDevice](https://link.jianshu.com?t=https:/developer.android.com/reference/android/bluetooth/BluetoothDevice.html)

代表一个远程蓝牙设备。这个类可以让你连接所代表的蓝牙设备或者获取一些有关它的信息，例如它的名字，地址和绑定状态等等。

[BluetoothGatt](https://link.jianshu.com?t=https:/developer.android.com/reference/android/bluetooth/BluetoothGatt.html)

这个类提供了 Bluetooth GATT 的基本功能。例如重新连接蓝牙设备，发现蓝牙设备的 Service 等等。

[BluetoothGattService](https://link.jianshu.com?t=https:/developer.android.com/reference/android/bluetooth/BluetoothGattService.html)

这一个类通过 BluetoothGatt#getService 获得，如果当前服务不可见那么将返回一个 null。这一个类对应上面说过的 Service。我们可以通过这个类的 getCharacteristic(UUID uuid) 进一步获取 Characteristic 实现蓝牙数据的双向传输。

[BluetoothGattCharacteristic](https://link.jianshu.com?t=https:/developer.android.com/reference/android/bluetooth/BluetoothGattCharacteristic.html)

这个类对应上面提到的 Characteristic。通过这个类定义需要往外围设备写入的数据和读取外围设备发送过来的数据。

**Android 蓝牙开发示例**

**第一步：声明所需要的权限**

<uses-permission android:name="android.permission.BLUETOOTH"/> 使用蓝牙所需要的权限

<uses-permission android:name="android.permission.BLUETOOTH\_ADMIN"/> 使用扫描和设置蓝牙的权限（申明这一个权限必须申明上面一个权限）

在Android5.0之前，是默认申请GPS硬件功能的。而在Android 5.0 之后，需要在manifest 中申明GPS硬件模块功能的使用。

<!-- Needed only if your app targets Android 5.0 (API level 21) or higher. -->

<uses-feature android:name="android.hardware.location.gps" />

在 Android 6.0 及以上，还需要打开位置权限。如果应用没有位置权限，蓝牙扫描功能不能使用（其它蓝牙操作例如连接蓝牙设备和写入数据不受影响）。

<uses-permission android:name="android.permission.ACCESS\_COARSE\_LOCATION"/>

**第二步：连接蓝牙前的初始化工作**

在建立蓝牙连接之前，需要确认设备支持 BLE。如果支持，再确认蓝牙是否开启。如果蓝牙没有开启，可以使用 BLuetoothAdapter 类来开启蓝牙。

1. 获取 BluetoothAdapter

private BluetoothAdapter mBluetoothAdapter;

// Initializes Bluetooth adapter.

final BluetoothManager bluetoothManager =

(BluetoothManager) getSystemService(Context.BLUETOOTH\_SERVICE);

mBluetoothAdapter = bluetoothManager.getAdapter();

1. 如果检测到蓝牙没有开启，尝试开启蓝牙

// Ensures Bluetooth is available on the device and it is enabled. If not,

// displays a dialog requesting user permission to enable Bluetooth.

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

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

startActivityForResult(enableBtIntent, REQUEST\_ENABLE\_BT);

}

**第三步：扫描蓝牙设备**

外围设备开启蓝牙后，会广播出许多的关于该设备的数据信息，例如 mac 地址，uuid 等等。通过这些数据我们可以筛选出需要的设备。

在 BluetoothAdapter 中，我们可以看到有两个扫描蓝牙的方法。第一个方法可以指定只扫描含有特定 UUID Service 的蓝牙设备，第二个方法则是扫描全部蓝牙设备。

boolean startLeScan(UUID[] serviceUuids, BluetoothAdapter.LeScanCallback callback)

boolean startLeScan(BluetoothAdapter.LeScanCallback callback)

**开启蓝牙扫描**

final BluetoothAdapter.LeScanCallback callback = new BluetoothAdapter.LeScanCallback() {

@Override

public void onLeScan(final BluetoothDevice device, int rssi, byte[] scanRecord) {

bluetoothDeviceArrayList.add(device);

Log.d(TAG, "run: scanning...");

}

};

mBluetoothAdapter.startLeScan(callback);

在 LeScanCallback 回调的方法中，第一个参数是代表蓝牙设备的类，可以通过这个类建立蓝牙连接获取关于这一个设备的一系列详细的参数，例如名字，MAC 地址等等；第二个参数是蓝牙的信号强弱指标，通过蓝牙的信号指标，我们可以大概计算出蓝牙设备离手机的距离。计算公式为：d = 10^((abs(RSSI) - A) / (10 \* n))；第三个参数是蓝牙广播出来的广告数据。

当执行上面的代码之后，一旦发现蓝牙设备，LeScanCallback 就会被回调，直到 stopLeScan 被调用。出现在回调中的设备会重复出现，所以如果我们需要通过 BluetoothDevice 获取外围设备的地址手动过滤掉已经发现的外围设备。

**停止蓝牙扫描**

void stopLeScan(BluetoothAdapter.LeScanCallback callback)

通过调用 BluetoothAdapter#stopLeScan 可以停止正在进行的蓝牙扫描。这里需要注意的是，传入的回调必须是开启蓝牙扫描时传入的回调，否则蓝牙扫描不会停止。

由于蓝牙扫描的操作比较消耗手机的能量。所以我们不能一直开着蓝牙，必须设置一段时间之后关闭蓝牙扫描。示例代码如下：

private void scanLeDevice(final boolean enable) {

if (enable) {

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

// 预先定义停止蓝牙扫描的时间（因为蓝牙扫描需要消耗较多的电量）

mHandler.postDelayed(new Runnable() {

@Override

public void run() {

mScanning = false;

mBluetoothAdapter.stopLeScan(mLeScanCallback);

}

}, SCAN\_PERIOD);

mScanning = true;

// 定义一个回调接口供扫描结束处理

mBluetoothAdapter.startLeScan(mLeScanCallback);

} else {

mScanning = false;

mBluetoothAdapter.stopLeScan(mLeScanCallback);

}

}

**第四步：连接蓝牙设备**

连接蓝牙设备可以通过 BluetoothDevice#ConnectGatt方法连接，也可以通过 BluetoothGatt#connect方法进行重新连接。以下分别是两个方法的官方说明：

BluetoothDevice#connectGatt

BluetoothGatt connectGatt(Context context, boolean autoConnect, BluetoothGattCallback callback)

第二个参数表示是否需要自动连接。如果设置为 true, 表示如果设备断开了，会不断的尝试自动连接。设置为 false 表示只进行一次连接尝试。

第三个参数是连接后进行的一系列操作的回调，例如连接和断开连接的回调，发现服务的回调，成功写入数据，成功读取数据的回调等等。

BluetoothGatt#connect

boolean connect()

调用这一个方法相当与调用 BluetoothDevice#connectGatt 且第二个参数 autoConnect 设置为 true。

当调用蓝牙的连接方法之后，蓝牙会异步执行蓝牙连接的操作，如果连接成功会回调 BluetoothGattCalbackl#onConnectionStateChange 方法。这个方法运行的线程是一个 Binder 线程，所以不建议直接在这个线程处理耗时的任务，因为这可能导致蓝牙相关的线程被阻塞。

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

这一个方法有三个参数，第一个就蓝牙设备的 Gatt 服务连接类。

第二个参数代表是否成功执行了连接操作，如果为 BluetoothGatt.GATT\_SUCCESS 表示成功执行连接操作，第三个参数才有效，否则说明这次连接尝试不成功。有时候，我们会遇到 status == 133 的情况，根据网上大部分人的说法，这是因为 Android 最多支持连接 6 到 7 个左右的蓝牙设备，如果超出了这个数量就无法再连接了。所以当我们断开蓝牙设备的连接时，还必须调用 BluetoothGatt#close 方法释放连接资源。否则，在多次尝试连接蓝牙设备之后很快就会超出这一个限制，导致出现这一个错误再也无法连接蓝牙设备。

第三个参数代表当前设备的连接状态，如果 newState == BluetoothProfile.STATE\_CONNECTED 说明设备已经连接，可以进行下一步的操作了（发现蓝牙服务，也就是 Service）。当蓝牙设备断开连接时，这一个方法也会被回调，其中的 newState == BluetoothProfile.STATE\_DISCONNECTED。

**第五步：发现服务**

在成功连接到蓝牙设备之后才能进行这一个步骤，也就是说在 BluetoothGattCalbackl#onConnectionStateChang 方法被成功回调且表示成功连接之后调用 BluetoothGatt#discoverService 这一个方法。当这一个方法被调用之后，系统会异步执行发现服务的过程，直到 BluetoothGattCallback#onServicesDiscovered 被系统回调之后，手机设备和蓝牙设备才算是真正建立了可通信的连接。

**到这一步，我们已经成功和蓝牙设备建立了可通信的连接，接下来就可以执行相应的蓝牙通信操作了，例如写入数据，读取蓝牙设备的数据等等。**

**读取数据**

当我们发现服务之后就可以通过 BluetoothGatt#getService 获取 BluetoothGattService，接着通过 BluetoothGattService#getCharactristic 获取 BluetoothGattCharactristic。

通过 BluetoothGattCharactristic#readCharacteristic 方法可以通知系统去读取特定的数据。如果系统读取到了蓝牙设备发送过来的数据就会调用 BluetoothGattCallback#onCharacteristicRead 方法。通过 BluetoothGattCharacteristic#getValue 可以读取到蓝牙设备的数据。以下是代码示例：

@Override

public void onCharacteristicRead(final BluetoothGatt gatt,

final BluetoothGattCharacteristic characteristic,

final int status) {

Log.d(TAG, "callback characteristic read status " + status

+ " in thread " + Thread.currentThread());

if (status == BluetoothGatt.GATT\_SUCCESS) {

Log.d(TAG, "read value: " + characteristic.getValue());

}

}

// 读取数据

BluetoothGattService service = gattt.getService(SERVICE\_UUID);

BluetoothGattCharacteristic characteristic = gatt.getCharacteristic(CHARACTER\_UUID);

gatt.readCharacteristic();

**写入数据**

和读取数据一样，在执行写入数据前需要获取到 BluetoothGattCharactristic。接着执行一下步骤：

1. 调用 BluetoothGattCharactristic#setValue 传入需要写入的数据（蓝牙最多单次1支持 20 个字节数据的传输，如果需要传输的数据大于这一个字节则需要分包传输）。
2. 调用 BluetoothGattCharactristic#writeCharacteristic 方法通知系统异步往设备写入数据。
3. 系统回调 BluetoothGattCallback#onCharacteristicWrite 方法通知数据已经完成写入。此时，我们需要执行 BluetoothGattCharactristic#getValue 方法检查一下写入的数据是否我们需要发送的数据，如果不是按照项目的需要判断是否需要重发。
4. 以下是示例代码：

@Override

public void onCharacteristicWrite(final BluetoothGatt gatt,

final BluetoothGattCharacteristic characteristic,

final int status) {

Log.d(TAG, "callback characteristic write in thread " + Thread.currentThread());

if(!characteristic.getValue().equal(sendValue)) {

// 执行重发策略

gatt.writeCharacteristic(characteristic);

}

}

//往蓝牙数据通道的写入数据

BluetoothGattService service = gattt.getService(SERVICE\_UUID);

BluetoothGattCharacteristic characteristic = gatt.getCharacteristic(CHARACTER\_UUID);

characteristic.setValue(sendValue);

gatt.writeCharacteristic(characteristic);

**向蓝牙设备注册监听实现实时读取蓝牙设备的数据**

BLE app通常需要获取设备中characteristic 变化的通知。下面的代码演示了怎么为一个Characteristic 设置一个监听。

mBluetoothGatt.setCharacteristicNotification(characteristic, enabled);

BluetoothGattDescriptor descriptor = characteristic.getDescriptor(

UUID.fromString(SampleGattAttributes.CLIENT\_CHARACTERISTIC\_CONFIG));

descriptor.setValue(BluetoothGattDescriptor.ENABLE\_NOTIFICATION\_VALUE);

mBluetoothGatt.writeDescriptor(descriptor);

值得注意的是，除了通过 BluetoothGatt#setCharacteristicNotification 开启 Android 端接收通知的开关，还需要往 Characteristic 的 Descriptor 属性写入开启通知的数据开关使得当硬件的数据改变时，主动往手机发送数据。

**最后一步：断开连接**

当我们连接蓝牙设备完成一系列的蓝牙操作之后就可以断开蓝牙设备的连接了。通过 BluetoothGatt#disconnect 可以断开正在连接的蓝牙设备。当这一个方法被调用之后，系统会异步回调 BluetoothGattCallback#onConnectionStateChange 方法。通过这个方法的 newState 参数可以判断是连接成功还是断开成功的回调。

由于 Android 蓝牙连接设备的资源有限，当我们执行断开蓝牙操作之后必须执行 BluetoothGatt#close 方法释放资源。需要注意的是通过 BluetoothGatt#close 方法也可以执行断开蓝牙的操作，不过 BluetoothGattCallback#onConnectionStateChange 将不会收到任何回调。此时如果执行 BluetoothGatt#connect 方法会得到一个蓝牙 API 的空指针异常。所以，我们推荐的写法是当蓝牙成功连接之后，通过 BluetoothGatt#disconnect 断开蓝牙的连接，紧接着在 BluetoothGattCallback#onConnectionStateChange 执行 BluetoothGatt#close 方法释放资源。

以下是代码示例：

@Override

public void onConnectionStateChange(final BluetoothGatt gatt, final int status,

final int newState) {

Log.d(TAG, "onConnectionStateChange: thread "

+ Thread.currentThread() + " status " + newState);

if (status != BluetoothGatt.GATT\_SUCCESS) {

String err = "Cannot connect device with error status: " + status;

// 当尝试连接失败的时候调用 disconnect 方法是不会引起这个方法回调的，所以这里

// 直接回调就可以了。

gatt.close();

Log.e(TAG, err);

return;

}

if (newState == BluetoothProfile.STATE\_CONNECTED) {

gatt.discoverService();

} else if (newState == BluetoothProfile.STATE\_DISCONNECTED) {

gatt.close();

}

}

**蓝牙操作的注意事项**

1. 蓝牙的写入操作( 包括 Descriptor 的写入操作), 读取操作必须序列化进行. 写入数据和读取数据是不能同时进行的, 如果调用了写入数据的方法, 马上调用又调用写入数据或者读取数据的方法,第二次调用的方法会立即返回 false, 代表当前无法进行操作. 详情可以参考 [蓝牙读写操作返回 false，为什么多次读写只有一次回调？](https://link.jianshu.com?t=http:/a1anwang.com/post-18.html)
2. Android 连接外围设备的数量有限，当不需要连接蓝牙设备的时候，必须调用 BluetoothGatt#close 方法释放资源。详细的参考可以看这里 [Android BLE 蓝牙开发的各种坑](https://link.jianshu.com?t=http:/www.eoeandroid.com/thread-902192-1-1.html?_dsign=d85c38ec)
3. 蓝牙 API 连接蓝牙设备的超时时间大概在 20s 左右，具体时间看系统实现。有时候某些设备进行蓝牙连接的时间会很长，大概十多秒。如果自己手动设置了连接超时时间（例如通过 Handler#postDelay 设置了 5s 后没有进入 BluetoothGattCallback#onConnectionStateChange 就执行 BluetoothGatt#close 操作强制释放断开连接释放资源）在某些设备上可能会导致接下来几次的连接尝试都会在 BluetoothGattCallback#onConnectionStateChange 返回 state == 133。另外可以参考这篇吐槽 [Android 中 BLE 连接出现“BluetoothGatt status 133”的解决方法](https://link.jianshu.com?t=http:/www.loverobots.cn/android-ble-connection-solution-bluetoothgatt-status-133.html)
4. 所有的蓝牙操作使用 Handler 固定在一条线程操作，这样能省去很多因为线程不同步导致的麻烦