蓝牙

传统蓝牙：比较耗电

低功耗蓝牙：

> 从蓝牙4.0开始包含两个蓝牙芯片模块：传统/经典蓝牙模块(Classic Bluetooth,简称BT)和低功耗蓝牙(Bluetooth Low Energy,简称BLE)
>
> 经典蓝牙是在之前的蓝牙1.0,1.2,2.0+EDR,2.1+EDR,3.0+EDR等基础上发展和完善起来的, 而低功耗蓝牙是Nokia的Wibree标准上发展起来的，是完全不同两个标准。
>
> 1. 经典蓝牙模块(BT)
>
> 泛指蓝牙4.0以下的模块，一般用于数据量比较大的传输，如：语音、音乐、较高数据量传输等。
>
> 经典蓝牙模块可再细分为：传统蓝牙模块和高速蓝牙模块。
>
> - 传统蓝牙模块在2004年推出，主要代表是支持蓝牙2.1协议的模块，在智能手机爆发的时期得到广泛支持。
> - 高速蓝牙模块在2009年推出，速率提高到约24Mbps，是传统蓝牙模块的八倍。
>
> 传统蓝牙有3个功率级别，Class1,Class2,Class3,分别支持100m,10m,1m的传输距离
>
> 2. 低功耗蓝牙模块(BLE)
>
> 泛指蓝牙4.0或更高的模块，蓝牙低功耗技术是低成本、短距离、可互操作的鲁棒性无线技术，工作在免许可的2.4GHz ISM射频频段。旨在提供显著降低的功耗； 可与功率要求更严格的 BLE 设备（例如近程传感器、心率监测仪和健身设备）通信
>
> 因为BLE技术采用非常快速的连接方式，因此平时可以处于“非连接”状态（节省能源），此时链路两端相互间只是知晓对方，只有在必要时才开启链路，然后在尽可能短的时间内关闭链路(每次最多传输20字节)。
>
> 低功耗蓝牙无功率级别，一般发送功率在7dBm，一般在空旷距离，达到20m应该是没有问题
>
> **注意：当用户使用 BLE 将其设备与其他设备配对时，用户设备上的所有应用都可以访问在这两个设备间传输的数据。因此，如果您的应用捕获敏感数据，您应实现应用层安全以保护此类数据的私密性**

Android手机蓝牙4.x都是双模蓝牙(既有经典蓝牙也有低功耗蓝牙)，而某些蓝牙设备为了省电是单模(只支持低功耗蓝牙)

- 经典蓝牙：
    1. 传声音

如蓝牙耳机、蓝牙音箱。蓝牙设计的时候就是为了传声音的，所以是近距离的音频传输的不二选择。

现在也有基于WIFI的音频传输方案，例如Airplay等，但是WIFI功耗比蓝牙大很多，设备无法做到便携。

因此固定的音响有WIFI的，移动的如耳机、便携音箱清一色都是基于经典蓝牙协议的。

2. 传大量数据

   例如某些工控场景，使用Android或Linux主控，外挂蓝牙遥控设备的，

   可以使用经典蓝牙里的SPP协议，当作一个无线串口使用。速度比BLE传输快多了。

   这里要注意的是，iPhone没有开放

- BLE蓝牙:

  耗电低，数据量小，如遥控类(鼠标、键盘)，传感设备(心跳带、血压计、温度传感器、共享单车锁、智能锁、防丢器、室内定位)

  是目前手机和智能硬件通信的性价比最高的手段，直线距离约50米，一节5号电池能用一年，传输模组成本10块钱，远比WIFI、4G等大数据量的通信协议更实用。

  虽然蓝牙距离近了点，但胜在直连手机，价格超便宜。以室内定位为例，商场每家门店挂个蓝牙beacon，就可以对手机做到精度10米级的室内定位，一个beacon的价格也就几十块钱而已

- 双模蓝牙:

  如智能电视遥控器、降噪耳机等。很多智能电视配的遥控器带有语音识别，需要用经典蓝牙才能传输声音。

  而如果做复杂的按键，例如原本键盘表上没有的功能，经典蓝牙的HID按键协议就不行了，得用BLE做私有协议。

  包括很多降噪耳机上通过APP来调节降噪效果，也是通过BLE来实现的私有通信协议

使用

Client

- 权限申请

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

<!-- If your app targets Android 9 or lower, you can declare
       ACCESS_COARSE_LOCATION instead. -->
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"
/>
```

- 初始化，打开蓝牙，扫描

```
val intentFilter = IntentFilter().apply {
    addAction(BluetoothDevice.ACTION_FOUND)
```

```
    addAction(BluetoothAdapter.ACTION_DISCOVERY_FINISHED)
    }
    registerReceiver(mReceiver, intentFilter)

    mBluetoothAdapter = BluetoothAdapter.getDefaultAdapter()
    mBluetoothAdapter?.let { bluetoothAdapter ->
        if (!bluetoothAdapter.isEnabled) {
            // enable bluetooth
            val enableBtIntent =
Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE
startActivityForResult(enableBtIntent, REQUEST_CODE_ENABLE_BT)
     }
       if (bluetoothAdapter.isDiscovering) {
            bluetoothAdapter.cancelDiscovery()
       }
       val success = bluetoothAdapter.startDiscovery()
       Log.d(localClassName, "connectionStart discovery:$success")
       // 已经配对的
       val pairedDevices: Set<BluetoothDevice>? =
bluetoothAdapter.bondedDevices
       if (pairedDevices.isNullOrEmpty()) {
            mPairedDevicesArrayAdapter?.add("no devices")
       } else {
            pairedDevices.forEach { device ->
                tv_title_paired.visibility = View.VISIBLE
                val deviceName = device.name
                val deviceHardwareAddress = device.address

 mPairedDevicesArrayAdapter?.add("$deviceName\n$deviceHardwareAddress")
                Log.d(localClassName, "for name:$deviceName.
address:$deviceHardwareAddress")
                if (deviceHardwareAddress == MAC_ADDRESS) {
                    if (mClientThread == null) {
                        mClientThread = BluetoothClientThread(this,
device)
                        mClientThread?.start()
                    }
                }
            }
            mPairedDevicesArrayAdapter?.notifyDataSetChanged()
        }
    }
    // 设置可检测
    // val discoverableIntent: Intent =
Intent(BluetoothAdapter.ACTION_REQUEST_DISCOVERABLE).apply {
    //     putExtra(BluetoothAdapter.EXTRA_DISCOVERABLE_DURATION, 300)
    // }
    // startActivity(discoverableIntent)
}
```

- 扫描结果

```
private val mReceiver = object : BroadcastReceiver() {
    override fun onReceive(context: Context?, intent: Intent?) {
        intent?.action?.let { action ->
```

```
            //attention the location permission to bluetooth
            when(action) {
                BluetoothDevice.ACTION_FOUND -> {
                    val device: BluetoothDevice? =
intent.getParcelableExtra(BluetoothDevice.EXTRA_DEVICE)
                    device?.apply {
                        val deviceName = name
                        val deviceAddress = address
                        Log.d(localClassName, "onReceive name:
$deviceName address: $deviceAddress")
                        if (bondState != BluetoothDevice.BOND_BONDED) {
                            if (address == MAC_ADDRESS) {

 mNewDevicesArrayAdapter?.insert("$deviceName\n$deviceAddress", 0)

                            } else {

mNewDevicesArrayAdapter?.add("$deviceName\n$deviceAddress")

                            }
                        }
                    }
                    mNewDevicesArrayAdapter?.notifyDataSetChanged()
                }
                BluetoothAdapter.ACTION_DISCOVERY_FINISHED -> {
                    title = "select to connect"
                    mNewDevicesArrayAdapter?.let {
                        if (it.count == 0) {
                            it.add("no devices")
                        }
                    }
                }
                else -> { }
            }
        }
    }
}
```

- 连接线程

```
private var mSocket: BluetoothSocket? =
device.createRfcommSocketToServiceRecord(UUID.fromString(BLUETOOTH_UUID
))

override fun run() {
    try {
        mSocket?.connect()
        mConnectedThread = mSocket?.let { ConnectedThread(it) }
        mConnectedThread?.start()
    } catch (e: IOException) {
        Log.e(javaClass.name, "connect error", e)
    }
}
```

- 读写线程

```
private class ConnectedThread(private val socket: BluetoothSocket) :
Thread() {
    private val mBuffer = ByteArray(1024)
    private val mDataOS = DataOutputStream(socket.outputStream)
    private val mDataIS = DataInputStream(socket.inputStream)

    init {
        Log.d(javaClass.name, "connected thread init $socket")
    }

    override fun run() {
        while (true) {
            try {
                val num = mDataIS.read(mBuffer)
                Log.d(javaClass.name, "read: ${String(mBuffer)}")
            } catch (e: IOException) {
                Log.d(javaClass.name, "read error", e)
            }
        }
    }

    fun writeFile(context: Context, fileUri: String) {
        var len: Int
        var fileIS: FileInputStream? = null
        val inputStream: InputStream?
        Log.d(javaClass.name, "client socket: ${socket.isConnected}")
        try {
            inputStream =
context.contentResolver.openInputStream(Uri.parse(fileUri))
            mDataOS.writeInt(FILE_SEND)
            inputStream?.let {
                mDataOS.writeInt(it.available())
                while (true) {
                    len = it.read(mBuffer)
                    if (len == -1) {
                        break
                    }
                Log.d(javaClass.name, "write ${mBuffer.size}
${mBuffer.count()} ${mBuffer.lastIndex} $len")
                    mDataOS.write(mBuffer, 0, len)
                }
                it.close()
                Log.d(javaClass.name, "Client data written ")
            }
        } catch (e: IOException) {
            Log.e(javaClass.name, "client socket error", e)
        } finally {
            fileIS?.close()
        }
    }
}
```

Server

- 权限申请
- 打开蓝牙，保证可检测

```
mBluetoothAdapter = BluetoothAdapter.getDefaultAdapter()
if (mBluetoothAdapter == null) {
    Log.e(localClassName, "can't apply bluetooth")
    return
}
mBluetoothAdapter?.enable()
// 设置可检测
// val discoverableIntent: Intent =
Intent(BluetoothAdapter.ACTION_REQUEST_DISCOVERABLE).apply {
//     putExtra(BluetoothAdapter.EXTRA_DISCOVERABLE_DURATION, 0)
// }
// startActivity(discoverableIntent)
mBluetoothAdapter?.name = "极光249"
Log.d(localClassName, "enable: ${mBluetoothAdapter?.isEnabled} name:
${mBluetoothAdapter?.name} mac: ${mBluetoothAdapter?.address}")
bindService(Intent(this, BluetoothServerService::class.java),
mConnection, Context.BIND_AUTO_CREATE)
```

- 启动服务线程

```
private inner class AcceptThread : Thread() {
    private val mServerSocket: BluetoothServerSocket? by
lazy(LazyThreadSafetyMode.NONE) {

 mBluetoothAdapter?.listenUsingInsecureRfcommWithServiceRecord(NAME,
UUID.fromString(BLUETOOTH_UUID))
    }

    override fun run() {
        Log.d(javaClass.name, "AcceptThread begin")
        name = "AcceptThread"
        try {
            val clientSocket: BluetoothSocket? =
mServerSocket?.accept()
            mHandler?.takeIf { mFilePath != null }?.apply {
                if (mServerThread == null) {
                    mServerThread = clientSocket?.let {
ServerThread(mHandler!!, mFilePath!!, it) }
                    mServerThread?.start()
                }
                cancel()
            }
        } catch (e: IOException) {
            Log.e(javaClass.name, "error", e)
            cancel()
        }
    }

    fun cancel() {
        try {
            mServerSocket?.close()
        } catch (e: IOException) {
```

```
            Log.e(javaClass.name, "Could not close the server mSocket", e)
        }
    }
}
```

- 读写线程

```
private class ServerThread : Thread {

    private var mHandler: Handler
    private var mFilePath: String
    private var mSocket: BluetoothSocket
    private val mDataIS: DataInputStream
    private val mDataOS: DataOutputStream
    private val mBuffer = ByteArray(1024)

    constructor(handler: Handler, filePath: String, socket: BluetoothSocket): this("ServerThread", handler, filePath, socket)

    constructor(name: String, handler: Handler, filePath: String, socket: BluetoothSocket) {
        this.name = name
        this.mHandler = handler
        this.mFilePath = filePath
        this.mSocket = socket
        mDataIS = DataInputStream(mSocket.inputStream)
        mDataOS = DataOutputStream(mSocket.outputStream)
    }

    override fun run() {
        var len: Int
        var fileOS: FileOutputStream? = null
        while (true) {
            try {
                Log.d(javaClass.name, "receive start")
                val tag = mDataIS.readInt()
                if (tag == FILE_SEND) {
                    val size = mDataIS.readInt()
                    var length = 0
                    val f = File(mFilePath, "bluetooth-shared-${System.currentTimeMillis()}.jpg")
                    val dirs = File(f.parent?:mFilePath)
                    dirs.takeIf { !it.exists() }?.apply { mkdirs() }
                    f.createNewFile()
                    fileOS = FileOutputStream(f)
                    while (true) {
                        len = mDataIS.read(mBuffer)
                        length += len
                        Log.d(javaClass.name, "size: $size receive $len")
                        if (len == -1) {
                            break
                        }
                        fileOS.write(mBuffer, 0, len)
```

```
                        // 记录接收的大小，在传输完成后跳出此次循环，避免一直阻塞在此
                        if (length >= size) {
                            break
                        }
                    }
                    Log.d(javaClass.name, "receive end: ${f.absolutePath} size: $size")
                    write("received ${f.path}".toByteArray())
                    val msg = Message.obtain()
                    msg.what = BluetoothServerActivity.MESSAGE_RECEIVE_FILE
                    msg.obj = f.absolutePath
                    mHandler.sendMessage(msg)
                }
                } catch (e: IOException) {
                    Log.e(javaClass.name, "readFile error", e)
                } finally {
                    fileOS?.close()
                }
            }
        }

        fun write(bytes: ByteArray) {
            val outputStream = mSocket.outputStream
            try {
                outputStream.write(bytes)
            } catch (e: IOException) {
                Log.e(javaClass.name, "writeFile error", e)
            }
        }

        fun cancle() {
            try {
                mSocket.close()
            } catch (e: IOException) {
                Log.e(javaClass.name, "close error", e)
            }
        }
    }
```

参考

Android-经典蓝牙(BT)-建立长连接传输短消息和文件