SerialPort

Shanya

2022 年 06 月 08 日

开始你的第一步

CHAPTER 1

安装

编辑 `Build.gradle` 文件并添加以下依赖项。

```
dependencies {
    implementation 'cn.shanyaliux.serialport:serialport:4.2.0'
}
```

如果你需要使用 `4.1.6` 及其以下版本，则按如下操作：

1. 添加 `JitPack` 仓库将 `JitPack` 存储库添加到您的构建文件中

```
allprojects {
    repositories {
        ...
        maven { url 'https://jitpack.io' }
    }
}
```

2. 添加依赖

```
dependencies {
    implementation 'com.gitee.Shanya:SerialPortSample:4.1.6'        //国内仓库
    implementation 'com.github.Shanyaliux:SerialPortSample:4.1.6'   //国外仓库
}
```

CHAPTER 2

快速上手 (Kotlin)

2.1 构建 SerialPort 实例

```
val serialPort = SerialPortBuilder.build(this)
```

2.2 搜索设备

使用方法 doDiscovery(context) 搜索设备:

```
serialPort.doDiscovery(this)
```

使用方法 getPairedDevicesListBD() 和 getUnPairedDevicesListBD() 获取搜索结果:

```
serialPort.getPairedDevicesListBD()                    //获取已配对设备列表
serialPort.getUnPairedDevicesListBD()          //获取未配对设备列表
```

2.3 连接设备

想要成功的连接设备，并且完成通信，设置正确的 UUID 是必不可少的一步。

2.3.1 设置传统设备 UUID

使用 `SerialPort` 的静态方法 `setLegacyUUID(uuid)` 设置传统设备的 UUID：

```
SerialPort.setLegacyUUID("00001101-0000-1000-8000-00805F9B34FB")
```

传统设备一**般**情况下，可以不用设置 UUID，使用默认的即可。

2.3.2 设置 BLE 设备 UUID

使用 `SerialPort` 的静态方法 `setBleUUID(uuid)` 设置 BLE 设备的 UUID：

```
SerialPort.setBleUUID("0000ffe1-0000-1000-8000-00805f9b34fb")
```

BLE 设备大多数情况下都需要设置 UUID，具体的 UUID 可以查询手册或咨询卖家。

除此之外，也可以使用方法 `printPossibleBleUUID()` 打印出可行的 UUID，自行选择尝试：

```
serialPort.printPossibleBleUUID()
```

2.3.3 建立连接

使用方法 `openDiscoveryActivity()` 打开内置的搜索页面选择设备进行连接：

```
serialPort.openDiscoveryActivity()
```

不想使用内置的搜索页面怎么办？

可以设置自定义的搜索页面或者直接使用设备地址进行连接。详见使用自定义的界面

2.4 接收消息

使用方法 `setReceivedDataCallback(receivedDataCallback)` 设置一个接收消息监听器：

```
serialPort.setReceivedDataCallback { data ->

        }
```

除此之外，你还可以在构建实例时配置监听器：

```
val serialPort = SerialPortBuilder
            .setReceivedDataCallback { data ->

            }
            .build(this)
```

2.5 发送消息

使用方法 sendData(data) 发送消息：

```
serialPort.sendData("Hello World")
```

至此，你已经可以快速的开发一款能够完成基本收发数据的串口应用了。当然，SerialPort 还有着更多的功能，请继续阅读说明文档。

CHAPTER 3

快速上手 (Java)

3.1 构建 SerialPort 实例

```
SerialPort serialPort = SerialPortBuilder.INSTANCE.build(this);
```

3.2 搜索设备

使用方法 doDiscovery(context) 搜索设备:

```
serialPort.doDiscovery(this);
```

使用方法 getPairedDevicesListBD() 和 getUnPairedDevicesListBD() 获取搜索结果:

```
serialPort.getPairedDevicesListBD();          //获取已配对设备列表
serialPort.getUnPairedDevicesListBD();          //获取未配对设备列表
```

如果搜索未结束，则可能获取的未配对设备列表为空或者不全.

3.3 连接设备

想要成功的连接设备，并且完成通信，设置正确的 UUID 是必不可少的一步。

3.3.1 设置传统设备 UUID

使用 SerialPort 的静态方法 setLegacyUUID(uuid) 设置传统设备的 UUID：

```
SerialPort.Companion.setLegacyUUID("00001101-0000-1000-8000-00805F9B34FB");
```

传统设备一**般**情况下，可以不用设置 UUID，使用默认的即可。

3.3.2 设置 BLE 设备 UUID

使用 SerialPort 的静态方法 setLegacyUUID(uuid) 设置 BLE 设备的 UUID：

```
SerialPort.Companion.setBleUUID("0000ffe1-0000-1000-8000-00805f9b34fb");
```

BLE 设备大多数情况下都需要设置 UUID，具体的 UUID 可以查询手册或咨询卖家。

除此之外，也可以使用方法 printPossibleBleUUID() 打印出可行的 UUID，自行选择尝试：

```
serialPort.printPossibleBleUUID()
```

3.3.3 建立连接

使用方法 openDiscoveryActivity() 打开内置的搜索页面选择设备进行连接：

```
serialPort.openDiscoveryActivity();
```

不想使用内置的搜索页面怎么办？

可以设置自定义的搜索页面或者直接使用设备地址进行连接。详见使用自定义的界面

3.4 接收消息

使用方法 setReceivedDataCallback(receivedDataCallback) 设置一个接收消息监听器：

```
serialPort.setReceivedDataCallback( (data) -> {

            return null;
        });
```

除此之外，你还可以在构建实例时配置监听器：

```
SerialPort serialPort = SerialPortBuilder.INSTANCE
                .setReceivedDataCallback( (data) -> {

                    return null;
                })
                .build(this);
```

3.5 发送消息

使用方法 sendData(data) 发送消息：

```
serialPort.sendData("Hello World");
```

至此，你已经可以快速的开发一款能够完成基本收发数据的串口应用了。当然，SerialPort 还有着更多的功能，请继续阅读说明文档。

CHAPTER 4

服务端

现在，可以构建 Android 服务端，实现两台 Android 之间的蓝牙通信。(目前仅支持单设备连接)

4.1 构建实例

```
val serialPortServer = SerialPortServerBuilder
            .setServerName("SerialPortServer")
            .setServerUUID("00001101-0000-1000-8000-00805F9B34FB")
            .setServerReceivedDataCallback {

            }
            .setServerConnectStatusCallback { status, bluetoothDevice ->

            }
            .build(this)
```

- `setServerName` 设置服务端名称
- `setServerUUID` 设置服务端 UUID，客户端连接时需设置传统设备的 UUID 与此相同
- `setServerReceivedDataCallback` 服务端接收消息监听
- `setServerConnectStatusCallback` 服务端连接状态监听
  - `status` 连接状态
  - `bluetoothDevice` 连接设备，当 `status` 为 false 则其为 null

4.2 打开服务

只有打开服务后，客户端才可以连接到服务端。

```
serialPortServer.openServer()
```

4.3 关闭服务

```
serialPortServer.closeServer()
```

4.4 设置服务端可发现状态

默认打开服务则会自动设置为可发现，关闭服务则设置为不可见。

```
setServerDiscoverable(status)
```

- `status` 为 `Boolean` 类型，表示可发现状态

4.5 断开连接

主动断开与客户端的连接

```
serialPortServer.disconnect()
```

4.6 发送消息

```
serialPortServer.sendData("Hello")
```

CHAPTER 5

配置

5.1 调试模式

在调试程序的时候，我们可以打开调试模式，这样就会打印各式各样的日志信息，在正式发布 APP 时关掉此开关即可减少资源的开销。设置方式如下：

```
val serialPort = SerialPortBuilder
            .isDebug(true)
            .build(this)
```

5.2 自动重连

5.2.1 启动时重连

开启此功能后，会在构建实例的时候执行一次自动重连，重连对象为上一次**成功连接**的设备。设置方式如下：

```
val serialPort = SerialPortBuilder
            .autoConnect(true)
            .build(this)
```

5.2.2 间隔自动重连

开启此功能后，会间隔一段时间自动重连一次（时间可自行设置），重连对象为上一次**成功连接**的设备。设置方式如下：

```
val serialPort = SerialPortBuilder
                        //第二个参数为间隔时间，若不指定则为默认 10000Ms
            .setAutoReconnectAtIntervals(true, 10000)
            .build(this)
```

5.3 忽略无名设备

开启此功能后，搜索设备时就会自动忽略设备名为空的设备。设置方式如下：

```
val serialPort = SerialPortBuilder
            .isIgnoreNoNameDevice(true)
            .build(this)
```

部分蓝牙设备可能会在第一次连接出现设备名为空的情况，请视情况而定开启此功能。

5.4 自动打开搜索界面

开启此功能后，在发送数据时，若发现未连接设备则会自动打开内置的搜索页面。设置方式如下：

```
val serialPort = SerialPortBuilder
            .autoOpenDiscoveryActivity(true)
            .build(this)
```

5.5 十六进制数据自动转换

开启此功能后，在收到的数据为十六进制时，会自动将其转换为字符串。设置方式如下：

```
val serialPort = SerialPortBuilder
            .autoHexStringToString(true)
            .build(this)
```

当然，你也可以使用方法 `hexStringToString(hexString)` 手动进行转换：

```
string = serialPort.hexStringToString(hexString)
```

5.6 内置搜索页面选择连接方式

开启此功能后在内置页面点击设备进行连接的时候，可以手动选择连接方式。但请注意若你的设备不支持你所选的连接方式，则连接不会成功。

```
val serialPort = SerialPortBuilder
            .setOpenConnectionTypeDialogFlag(true)
            .build(this)
```

5.7 配置器

配置可以将上述的多种配置一次性传入 SerialPortBuilder。

```
val config = SerialPortConfig()
val serialPort = SerialPortBuilder
            .setConfig(config)
            .build(this)
```

其中配置器可设置的参数如下表所示 (加粗的表示默认值):

其中关于 UUID 的设置注意事项参考: ble 设备设置 UUID

5.8 设置搜索时长

使用此方法配置搜索设备时长:

```
//参数为时间，单位毫秒
val serialPort = SerialPortBuilder
            .setDiscoveryTime(10000)
            .build(this)
```

CHAPTER 6

搜索和连接

6.1 内置的界面

为了更加方便快速的帮助开发串口通信应用程序，我们内部集成了一个必备的搜索和连接页面，使用方法 `openDiscoveryActivity()` 打开一个内置的界面：

```
serialPort.openDiscoveryActivity()
```

6.2 使用自定义的界面

当然了，在更多的情况我们的搜索和连接页面需要更加的美观和定制化。那么，可以使用方法 `serialPort.openDiscoveryActivity(intent)` 打开一个你自定义的页面：

```
//这里修改为你自定义的 Activity 即可
val intent = Intent(this,DiscoveryActivity::class.java)
serialPort.openDiscoveryActivity(intent)
```

6.3 搜索设备

6.3.1 开始搜索

使用方法 doDiscovery(context) 即可开始搜索设备：

```
serialPort.doDiscovery(this)
```

6.3.2 停止搜索

使用方法 cancelDiscovery(context) 即可开始搜索设备：

```
serialPort.cancelDiscovery(this)
```

6.3.3 搜索状态的监听

使用方法 setDiscoveryStatusWithTypeCallback(discoveryStatusWithTypeCallback) 或者 setDiscoveryStatusCallback(discoveryStatusCallback) 设置一个搜索状态监听器：

```
//status 为搜索状态
serialPort.setDiscoveryStatusCallback{ status ->

}
//搜索状态带类型的监听
//deviceType = SerialPort.DISCOVERY_BLE 搜索 BLE 设备
//deviceType = SerialPort.DISCOVERY_LEGACY 搜索传统类型
//status 为搜索状态
serialPort.setDiscoveryStatusWithTypeCallback { deviceType, status ->

}
```

除此之外，你还可以在构建实例时配置监听器：

```
//status 为搜索状态
val serialPort = SerialPortBuilder
            .setDiscoveryStatusCallback { status ->

            }
            .build(this)
//搜索状态带类型的监听
//deviceType = SerialPort.DISCOVERY_BLE 搜索 BLE 设备
//deviceType = SerialPort.DISCOVERY_LEGACY 搜索传统类型
```

(下页继续)

(续上页)

```
//status 为搜索状态
val serialPort = SerialPortBuilder
            .setDiscoveryStatusWithTypeCallback { deviceType, status ->

            }
            .build(this)
```

6.3.4 获取搜索结果

使用方法 getPairedDevicesListBD() 和 getUnPairedDevicesListBD() 获取搜索结果：

```
serialPort.getPairedDevicesListBD()                    //获取已配对设备列表
serialPort.getUnPairedDevicesListBD()          //获取未配对设备列表
```

如果搜索未结束，则可能获取的未配对设备列表为空或者不全。

6.4 连接设备

想要成功的连接设备，并且完成通信，设置正确的 UUID 是必不可少的一步。

6.4.1 传统设备

设置 UUID

使用 SerialPort 的静态方法 setLegacyUUID(uuid) 设置传统设备的 UUID：

```
SerialPort.setLegacyUUID("00001101-0000-1000-8000-00805F9B34FB")
```

传统设备一般情况下，可以不用设置 UUID，使用默认的即可。

建立连接

使用方法 connectLegacyDevice(address) 与传统设备建立连接：

```
serialPort.connectLegacyDevice("98:D3:32:21:67:D0")
```

6.4.2 BLE 设备

设置 UUID

使用 SerialPort 的静态方法 setBleUUID(uuid) 设置 BLE 设备的 UUID，或者使用 setBleSendUUID 和 setBleReadUUID 分别独立设置发送和接收的 UUID：

```
SerialPort.setBleUUID("0000ffe1-0000-1000-8000-00805f9b34fb")
SerialPort.setBleReadUUID("0000ffe1-0000-1000-8000-00805f9b34fb")
SerialPort.setBleSendUUID("0000ffe1-0000-1000-8000-00805f9b34fb")
```

如果独立设置了 UUID，则以独立设置的为准。

BLE 设备大多数情况下都需要设置 UUID，具体的 UUID 可以查询手册或咨询卖家。

除此之外，也可以使用方法 printPossibleBleUUID() 打印出可行的 UUID，详情见：打印 uuid 及其属性

建立连接

使用方法 connectBle(address) 与传统设备建立连接：

```
serialPort.connectBle("98:D3:32:21:67:D0")
```

6.4.3 断开连接

使用方法 disconnect() 与传统设备建立连接：

```
serialPort.disconnect()
```

6.4.4 连接状态的监听

使用方法 setConnectionStatusCallback(connectionStatusCallback) 设置一个连接状态的监听器：

```
serialPort.setConnectStatusCallback { status, bluetoothDevice ->


}
```

除此之外，你还可以在构建实例时配置监听器：

```
val serialPort = SerialPortBuilder
            .setConnectionStatusCallback { status, bluetoothDevice ->


            }
            .build(this)
```

这里的 bluetoothDevice 使用的时官方的类，其包含了蓝牙的设备的各种信息。详见官方文档

在之前版本使用的是自定义的 Device 类（不建议使用），其包含了：设备名、设备地址、设备类型。其实现如下：

```
@Deprecated("该类在 4.0.0 版本被弃用, 将直接使用官方的 BluetoothDevice 类代替")
data class Device(
    val name:String,
    val address:String,
    val type:Int = 0
)
```

6.4.5 BLE 可以工作回调

该回调在 BLE 设备连接成功，并且可以工作之后触发，可用于配置连接成功后自动发送消息。使用方法如下：

```
serialPort.setBleCanWorkCallback {

}
```

除此之外，你还可以在构建实例时配置监听器：

```
val serialPort = SerialPortBuilder
            .setBleCanWorkCallback {

                        }
            .build(this)
```

CHAPTER 7

收发数据

7.1 设置数据格式

使用方法 `setReadDataType(type)` 和 `setSendDataType(type)` 来设置手法数据的格式：

7.1.1 设置接收消息格式

```
//SerialPort.READ_HEX 十六进制
//SerialPort.READ_STRING 字符串
//不设置则默认字符串形式
serialPort.setReadDataType(SerialPort.READ_HEX)
```

除此之外，你还可以在构建实例时设置接收数据格式：

```
//SerialPort.READ_HEX 十六进制
//SerialPort.READ_STRING 字符串
//不设置则默认字符串形式
val serialPort = SerialPortBuilder
            .setReadDataType(SerialPort.READ_HEX)
            .build(this)
```

7.1.2 设置发送数据格式

```
//SerialPort.SEND_HEX 十六进制
//SerialPort.SEND_STRING 字符串
//不设置则默认字符串形式
serialPort.setSendDataType(SerialPort.SEND_HEX )
```

除此之外，你还可以在构建实例时设置接收数据格式：

```
//SerialPort.SEND_HEX 十六进制
//SerialPort.SEND_STRING 字符串
//不设置则默认字符串形式
val serialPort = SerialPortBuilder
            .setSendDataType(SerialPort.SEND_HEX)
            .build(this)
```

目前针对于 BLE 设备的数据收发暂不支持设置格式，仅支持字符串格式。如果实在需要十六进制的数据格式，暂时可以参考传统设备的处理方式自行实现。

参考代码链接：HexStringToString、StringToHex

7.2 接收消息

7.2.1 字符串和十六进制

使用方法 `setReceivedDataCallback(receivedDataCallback)` 设置一个接收消息监听器：

```
serialPort.setReceivedDataCallback { data ->

        }
```

除此之外，你还可以在构建实例时配置监听器：

```
val serialPort = SerialPortBuilder
            .setReceivedDataCallback { data ->

            }
            .build(this)
```

7.2.2 字节数组

在接收消息的时候，也可以选择获取**字节数组**，方法如下：

```
serialPort.setReceivedBytesCallback { bytes ->

        }
```

除此之外，你还可以在构建实例时配置监听器：

```
val serialPort = SerialPortBuilder
            .setReceivedBytesCallback { bytes ->

            }
            .build(this)
```

7.3 发送消息

使用方法 `sendData(data)` 发送消息：

7.3.1 字符串

```
serialPort.sendData("Hello World")
```

7.3.2 十六进制

```
serialPort.sendData("0C FF")
```

所有的十六进制应为**两位**，不足两位的前方补 0，不区分大小写。

7.3.3 BLE 设备发送字节数组

目前 BLE 设备支持直接发送字节数组

```
serialPort.sendData(bytes)
```

CHAPTER 8

Toast 提示设置

8.1 配置方法

```
//是否显示
SerialPortToast.connectSucceeded.status = true
//提示内容 content 是字符串 id
SerialPortToast.connectSucceeded.content = R.string.connectSucceededToast
//显示时长 Toast.LENGTH_SHORT 或 Toast.LENGTH_LONG
SerialPortToast.connectSucceeded.time = Toast.LENGTH_SHORT
```

8.2 可选配置项

CHAPTER 9

工具

9.1 打印 UUID 及其属性

若我们不能知晓当前 BLE 设备的 UUID 可以调用函数 `printPossibleBleUUID` 来打印出当前连接设备的可选 UUID

其中 `Properties` 为二进制数，其每一位对应的意思见下表：

9.2 字符串转换成十六进制

```
/**
 * 字符串转换成十六进制
 * @param str 待转换的字符串
 * @return 十六进制数组
 */
DataUtil.string2hex("Hello")
```

9.3 字节数组按要求的编码格式转换成字符串

```
/**
 * 字节数组按要求的编码格式转换成字符串
 * @param bytes 带转换的字节数组
 * @param charsetName 要求的编码格式
 * @return 转换成功的字符串
 */
SerialPortTools.bytes2string(bytes, "GBK")
```

9.4 字符串按要求的编码格式转换成字节数组

```
/**
 * 字符串按要求的编码格式转换成字节数组
 * @param string 带转换的字符串
 * @param charsetName 要求的编码格式
 * @return 转换成功的字节数组
 */
SerialPortTools.bytes2string("Hello", "GBK")
```

CHAPTER 10

配置

10.1 调试模式

在调试程序的时候，我们可以打开调试模式，这样就会打印各式各样的日志信息，在正式发布 APP 时关掉此开关即可减少资源的开销。设置方式如下：

```
SerialPort serialPort = SerialPortBuilder.INSTANCE
                .isDebug(true)
                .build(this);
```

10.2 自动重连

10.2.1 启动时重连

开启此功能后，会在构建实例的时候执行一次自动重连，重连对象为上一次**成功连接**的设备。设置方式如下：

```
SerialPort serialPort = SerialPortBuilder.INSTANCE
                .autoConnect(true)
                .build(this);
```

10.2.2 间隔自动重连

开启此功能后，会间隔一段时间自动重连一次（时间可自行设置），重连对象为上一次**成功连接**的设备。设置方式如下：

```
SerialPort serialPort = SerialPortBuilder.INSTANCE
                            //第二个参数为间隔时间，若不指定则为默认 10000Ms
                .setAutoReconnectAtIntervals(true, 10000)
                .build(this);
```

10.3 忽略无名设备

开启此功能后，搜索设备时就会自动忽略设备名为空的设备。设置方式如下：

```
SerialPort serialPort = SerialPortBuilder.INSTANCE
                .isIgnoreNoNameDevice(true)
                .build(this);
```

部分蓝牙设备可能会在第一次连接出现设备名为空的情况，请视情况而定开启此功能。

10.4 自动打开搜索界面

开启此功能后，在发送数据时，若发现未连接设备则会自动打开内置的搜索页面。设置方式如下：

```
SerialPort serialPort = SerialPortBuilder.INSTANCE
                .autoOpenDiscoveryActivity(true)
                .build(this);
```

10.5 十六进制数据自动转换

开启此功能后，在收到的数据为十六进制时，会自动将其转换为字符串。设置方式如下：

```
SerialPort serialPort = SerialPortBuilder.INSTANCE
                .autoHexStringToString(true)
                .build(this);
```

当然，你也可以使用方法 `hexStringToString(hexString)` 手动进行转换：

```
string = serialPort.hexStringToString(hexString)
```

10.6 内置搜索页面选择连接方式

开启此功能后在内置页面点击设备进行连接的时候，可以手动选择连接方式。但请注意若你的设备不支持你所选的连接方式，则连接不会成功。

```
SerialPort serialPort = SerialPortBuilder.INSTANCE
                .setOpenConnectionTypeDialogFlag(true)
                .build(this);
```

10.7 配置器

配置可以将上述的多种配置一次性传入 SerialPortBuilder。

```
SerialPortConfig config = new SerialPortConfig();
SerialPort serialPort = SerialPortBuilder.INSTANCE
                .setConfig(config)
                .build(this);
```

其中配置器可设置的参数如下表所示 (加粗的表示默认值):

其中关于 UUID 的设置注意事项参考: ble 设备设置 UUID

10.8 设置搜索时长

使用此方法配置搜索设备时长:

```
//参数为时间，单位毫秒
SerialPort serialPort = SerialPortBuilder.INSTANCE
                .setDiscoveryTime(10000)
                .build(this);
```

CHAPTER 11

搜索和连接

11.1 内置的界面

为了更加方便快速的帮助开发串口通信应用程序，我们内部集成了一个必备的搜索和连接页面，使用方法 `openDiscoveryActivity()` 打开一个内置的界面：

```
serialPort.openDiscoveryActivity();
```

11.2 使用自定义的界面

当然了，在更多的情况我们的搜索和连接页面需要更加的美观和定制化。那么，可以使用方法 `serialPort.openDiscoveryActivity(intent)` 打开一个你自定义的页面：

```
//这里修改为你自定义的 Activity 即可
Intent intent = new Intent(this, DiscoveryActivity.class);
serialPort.openDiscoveryActivity(intent);
```

11.3 搜索设备

11.3.1 开始搜索

使用方法 doDiscovery(context) 即可开始搜索设备：

```
serialPort.doDiscovery(this);
```

11.3.2 停止搜索

使用方法 cancelDiscovery(context) 即可开始搜索设备：

```
serialPort.cancelDiscovery(this);
```

11.3.3 搜索状态的监听

使用方法 setDiscoveryStatusWithTypeCallback(discoveryStatusWithTypeCallback) 或者 setDiscoveryStatusCallback(discoveryStatusCallback) 设置一个搜索状态监听器：

```
//status 为搜索状态
serialPort.setDiscoveryStatusCallback((status) ->{

   return null;
});
//搜索状态带类型的监听
//deviceType = SerialPort.DISCOVERY_BLE 搜索 BLE 设备
//deviceType = SerialPort.DISCOVERY_LEGACY 搜索传统类型
//status 为搜索状态
serialPort.setDiscoveryStatusWithTypeCallback((deviceType, status) -> {

return null;
});
```

除此之外，你还可以在构建实例时配置监听器：

```
//status 为搜索状态
SerialPort serialPort = SerialPortBuilder.INSTANCE
                .setDiscoveryStatusCallback( (status) -> {

                    return null;
                })
                .build(this);
```

(下页继续)

(续上页)

```
//搜索状态带类型的监听
//deviceType = SerialPort.DISCOVERY_BLE 搜索 BLE 设备
//deviceType = SerialPort.DISCOVERY_LEGACY 搜索传统类型
//status 为搜索状态
SerialPort serialPort = SerialPortBuilder.INSTANCE
                .setDiscoveryStatusWithTypeCallback( (deviceType, status) -> {

                    return null;
                })
                .build(this);
```

11.3.4 获取搜索结果

使用方法 `getPairedDevicesListBD()` 和 `getUnPairedDevicesListBD()` 获取搜索结果：

```
serialPort.getPairedDevicesListBD();          //获取已配对设备列表
serialPort.getUnPairedDevicesListBD();          //获取未配对设备列表
```

如果搜索未结束，则可能获取的未配对设备列表为空或者不全。

11.4 连接设备

想要成功的连接设备，并且完成通信，设置正确的 UUID 是必不可少的一步。

11.4.1 传统设备

设置 UUID

使用 `SerialPort` 的静态方法 `setLegacyUUID(uuid)` 设置传统设备的 UUID：

```
SerialPort.Companion.setLegacyUUID("00001101-0000-1000-8000-00805F9B34FB");
```

传统设备一般情况下，可以不用设置 UUID，使用默认的即可。

建立连接

使用方法 `connectLegacyDevice(address)` 与传统设备建立连接：

```
serialPort.connectLegacyDevice("98:D3:32:21:67:D0");
```

11.4.2 BLE 设备

设置 UUID

使用 `SerialPort` 的静态方法 `setBleUUID(uuid)` 设置 BLE 设备的 UUID，或者使用 `setBleSendUUID` 和 `setBleReadUUID` 分别独立设置发送和接收的 UUID：

```
SerialPort.Companion.setBleUUID("0000ffe1-0000-1000-8000-00805f9b34fb");
SerialPort.Companion.setBleReadUUID("0000ffe1-0000-1000-8000-00805f9b34fb");
SerialPort.Companion.setBleSendUUID("0000ffe1-0000-1000-8000-00805f9b34fb");
```

如果独立设置了 UUID，则以独立设置的为准。

BLE 设备大多数情况下都需要设置 UUID，具体的 UUID 可以查询手册或咨询卖家。

除此之外，也可以使用方法 `printPossibleBleUUID()` 打印出可行的 UUID，详情见：打印 uuid 及其属性

建立连接

使用方法 `connectBle(address)` 与传统设备建立连接：

```
serialPort.connectBle("98:D3:32:21:67:D0");
```

11.4.3 断开连接

使用方法 `disconnect()` 与传统设备建立连接：

```
serialPort.disconnect();
```

11.4.4 连接状态的监听

使用方法 setConnectionStatusCallback(connectionStatusCallback) 设置一个连接状态的监听器：

```
serialPort.setConnectionStatusCallback((status,bluetoothDevice)->{

        return null;
});
```

除此之外，你还可以在构建实例时配置监听器：

```
SerialPort serialPort = SerialPortBuilder.INSTANCE
                .setConnectionStatusCallback( (status, bluetoothDevice) -> {

                    return null;
                })
                .build(this);
```

这里的 bluetoothDevice 使用的时官方的类，其包含了蓝牙的设备的各种信息。详见官方文档

在之前版本使用的是自定义的 Device 类（不建议使用），其包含了：设备名、设备地址、设备类型。其实现如下：

```
@Deprecated("该类在 4.0.0 版本被弃用, 将直接使用官方的 BluetoothDevice 类代替")
data class Device(
    val name:String,
    val address:String,
    val type:Int = 0
)
```

11.4.5 BLE 可以工作回调

该回调在 BLE 设备连接成功，并且可以工作之后触发，可用于配置连接成功后自动发送消息。使用方法如下：

```
serialPort.setBleCanWorkCallback( () -> {

        return null;
        });
```

除此之外，你还可以在构建实例时配置监听器：

```
SerialPort serialPort = SerialPortBuilder.INSTANCE
        .setBleCanWorkCallback( () -> {

        return null;
    })
    .build(this);
```

CHAPTER 12

收发数据

12.1 设置数据格式

使用方法 `setReadDataType(type)` 和 `setSendDataType(type)` 来设置手法数据的格式：

12.1.1 设置接收消息格式

```
//SerialPort.READ_HEX 十六进制
//SerialPort.READ_STRING 字符串
//不设置则默认字符串形式
serialPort.setReadDataType(SerialPort.READ_HEX);
```

除此之外，你还可以在构建实例时设置接收数据格式：

```
//SerialPort.READ_HEX 十六进制
//SerialPort.READ_STRING 字符串
//不设置则默认字符串形式
SerialPort serialPort = SerialPortBuilder.INSTANCE
                .setReadDataType(SerialPort.READ_HEX)
                .build(this);
```

12.1.2 设置发送数据格式

```
//SerialPort.SEND_HEX 十六进制
//SerialPort.SEND_STRING 字符串
//不设置则默认字符串形式
serialPort.setSendDataType(SerialPort.SEND_HEX);
```

除此之外，你还可以在构建实例时设置接收数据格式：

```
//SerialPort.SEND_HEX 十六进制
//SerialPort.SEND_STRING 字符串
//不设置则默认字符串形式
SerialPort serialPort = SerialPortBuilder.INSTANCE
                .setSendDataType(SerialPort.SEND_HEX)
                .build(this);
```

目前针对于 BLE 设备的数据收发暂不支持设置格式，仅支持字符串格式。如果实在需要十六进制的数据格式，暂时可以参考传统设备的处理方式自行实现。

参考代码链接：HexStringToString、StringToHex

12.2 接收消息

12.2.1 字符串和十六进制

使用方法 `setReceivedDataCallback(receivedDataCallback)` 设置一个接收消息监听器：

```
serialPort.setReceivedDataCallback( (data) -> {

            return null;
        });
```

除此之外，你还可以在构建实例时配置监听器：

```
SerialPort serialPort = SerialPortBuilder.INSTANCE
                .setReceivedDataCallback( (data) -> {

                    return null;
                })
                .build(this);
```

12.2.2 字节数组

在接收消息的时候，也可以选择获取**字节数组**，方法如下：

```
serialPort.setReceivedBytesCallback( (bytes) -> {

            return null;
        });
```

除此之外，你还可以在构建实例时配置监听器：

```
SerialPort serialPort = SerialPortBuilder.INSTANCE
                .setReceivedBytesCallback( (bytes) -> {

                    return null;
                })
                .build(this);
```

12.3 发送消息

使用方法 `sendData(data)` 发送消息：

12.3.1 字符串

```
serialPort.sendData("Hello World");
```

12.3.2 十六进制

```
serialPort.sendData("0C FF");
```

所有的十六进制应为**两位**，不足两位的前方补 0，不区分大小写。

12.3.3 BLE 设备发送字节数组

目前 BLE 设备支持直接发送字节数组

```
serialPort.sendData(bytes);
```

CHAPTER 13

Toast 提示设置

13.1 配置方法

```
//是否显示
SerialPortToast.INSTANCE.getConnectSucceeded().setStatus(true);
//提示内容 content 是字符串 id
SerialPortToast.INSTANCE.getConnectSucceeded().setContent(R.string.
↪connectSucceededToast);
//显示时长 Toast.LENGTH_SHORT 或 Toast.LENGTH_LONG
SerialPortToast.INSTANCE.getConnectSucceeded().setTime(Toast.LENGTH_SHORT);
```

13.2 可选配置项

CHAPTER 14

工具

14.1 打印 UUID 及其属性

若我们不能知晓当前 BLE 设备的 UUID 可以调用函数 `printPossibleBleUUID` 来打印出当前连接设备的可选 UUID

其中 `Properties` 为二进制数，其每一位对应的意思见下表：

14.2 字符串转换成十六进制

```
/**
 * 字符串转换成十六进制
 * @param str 待转换的字符串
 * @return 十六进制数组
 */
DataUtil.INSTANCE.string2hex("Hello");
```

14.3 字节数组按要求的编码格式转换成字符串

```
/**
 * 字节数组按要求的编码格式转换成字符串
 * @param bytes 带转换的字节数组
 * @param charsetName 要求的编码格式
 * @return 转换成功的字符串
 */
SerialPortTools.bytes2string(bytes, "GBK");
```

14.4 字符串按要求的编码格式转换成字节数组

```
/**
 * 字符串按要求的编码格式转换成字节数组
 * @param string 带转换的字符串
 * @param charsetName 要求的编码格式
 * @return 转换成功的字节数组
 */
SerialPortTools.bytes2string("Hello", "GBK");
```

CHAPTER 15

有疑必看

15.1 功能支持

15.1.1 支持 BLE 设备吗？

支持，`SerialPort` 从 `4.0.0` 版本开始全面支持 BLE 设备。

15.1.2 有自动重连机制吗？

有，可以设置在启动时重连，也可以间隔时间自动重连。详见自动重连

15.2 常见问题

15.2.1 为什么自定义页面的可用设备列表为空？

在 Android6.0 之后一定要给上定位权限才可以搜索到可用设备。

15.2.2 为什么 BLE 设备连接成功了，不能收发消息，有时还是发出异常？

BLE 设备连接成功后，还需要设置正确的 UUID 才可以正常通信。具体设置方法见设置 BLE 设备 UUID

15.3 还有其他问题怎么解决？

15.3.1 开启调试模式

打开调试模式查看打印的日志信息。详见调试模式

15.3.2 利用强大的搜索引擎

15.3.3 加群

若通过以上方法仍未解决问题，请加入 QQ 技术交流群。

CHAPTER 16

更新日志

16.1 4.2.0 版本已经在 7/6/2022 发布:

- 修复一些关于蓝牙权限的编译告警
- 升级 `Kotlin` 和 `Gradle` 版本
- 标记 `ConnectionResultCallback` 过时
- 新增 Ble 设备发送字节数组
- 新增 Ble 设备可以工作回调
- 新增服务端配置

16.2 4.1.9 版本已经在 9/5/2022 发布:

- 修复当发送 UUID 设置错误时应用闪退

16.3 4.1.8 版本已经在 14/4/2022 发布:

- 修复 autoOpenDiscoveryActivity 始终为 true
- DiscoveryActivity 添加英文
- 添加 setDiscoveryTime

CHAPTER 17

赞助

SerialPort 是采用 Apache-2.0 许可的开源项目，使用完全免费，但您的赞助可以使 SerialPort 获得更健康稳定的发展。

如果您认可作者的努力或者 SerialPort 确实帮助到了您，请到 Gitee和 GitHub上点个 Star 鼓励一下吧。另外如果您口袋充裕，可以扫以下的赞助码，成为 SerialPort 的赞助人：

CHAPTER 18

English

CHAPTER 19

简体中文