密级：公开资料

Android BLE API

使用说明

V1.0

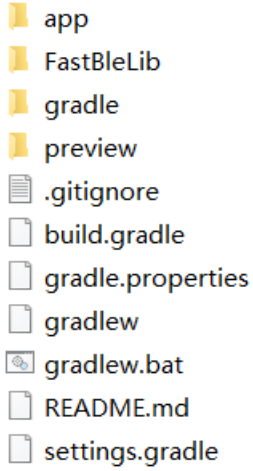
修订历史

版本信息管理

版本号	时间	更新记录	编辑者
V1.0	2019.01.19	初版发布	郭振兴

目录

1. Android BLE SDK 介绍



1） app 为示例代码
2） FastBleLib 为 BLE 库文件
3） gradle 开发环境配置
4） preview 示例截图

2. 代码示例

2.1 概述

Android Bluetooth Low Energy 蓝牙快速开发框架。使用回调方式处理：scan ,connect ,notify,indicate,write,read 等系列蓝牙操作。每一个 characteristic 会与个 callback 形成一对应的监听关系。

2.2 代码介绍

初始化

```
bleManager = new BleManager(this);
```

判断设备是否支持 BLE

```
bleManager.isSupportBle();
```

开启或关闭蓝牙

```
bleManager.enableBluetooth();
bleManager.disableBluetooth();
```

扫描出周围所有蓝牙可连接设备

可获得周围蓝牙设备 BluetoothDevice 对象数组

```
bleManager.scanDevice(new ListScanCallback(TIME_OUT) {
    @Override
    public void onLeScan(BluetoothDevice device, int rssi, byte[] scanRecord)
{
        super.onLeScan(device, rssi, scanRecord);
        Log.i(TAG, " 发现设备:  " + device.getName());
    }
    @Override
    public void onDeviceFound(BluetoothDevice[] devices) {
        Log.i(TAG, " 共发现 " + devices.length + " 台设备 ");
        for (int i = 0; i < devices.length; i++) {
            Log.i(TAG, "name:" + devices[i].getName() + "------mac:" + devices
[i].getAddress());
        }
    }
});
```

直连某一个设备

当搜索到周围设备之后，可以选择选择某一个设备和其连接，传入的参数即为这个

BluetoothDevice 对象

```
bleManager.connectDevice(sampleDevice, new BleGattCallback() {
    @Override
    public void onNotFoundDevice() {
```

```
        Log.i(TAG, " 未发现设备！ ");
    }
    @Override
    public void onFoundDevice(BluetoothDevice device) {
        Log.i(TAG, " 发现设备 ： " + device.getAddress());
    }
    @Override
    public void onConnectSuccess(BluetoothGatt gatt, int status) {
        Log.i(TAG, " 连接成功！ ");
        gatt.discoverServices();
    }
    @Override
    public void onServicesDiscovered(BluetoothGatt gatt, int status) {
        Log.i(TAG, " 服务被发现！ ");
        bleManager.getBluetoothState();
    }
    @Override
    public void onConnectFailure(BleException exception) {
        Log.i(TAG, " 连接失败或连接中断： " + exception.toString());
        bleManager.handleException(exception);
    }
});
```

扫描指定名称的设备、并连接

如果你确定周围有已知名称的蓝牙设备，或只需要连接指定名称的蓝牙设备，而忽略其他名称的设备，

```
bleManager.scanNameAndConnect(
        DEVICE_NAME,
        TIME_OUT,
        new BleGattCallback() {
            @Override
            public void onNotFoundDevice() {
                Log.i(TAG, " 未发现设备！ ");
            }
            @Override
            public void onFoundDevice(BluetoothDevice device) {
                Log.i(TAG, " 发现设备 ： " + device.getAddress());
            }
            @Override
            public void onConnectSuccess(BluetoothGatt gatt, int status) {
```

```
            Log.i(TAG, " 连接成功！ ");
            gatt.discoverServices();
        }
        @Override
        public void onServicesDiscovered(BluetoothGatt gatt, int status) {
            Log.i(TAG, " 服务被发现！ ");
            bleManager.getBluetoothState();
        }
        @Override
        public void onConnectFailure(BleException exception) {
            Log.i(TAG, " 连接失败或连接中断： " + exception.toString());
            bleManager.handleException(exception);
        }
    });
```

扫描指定 MAC 地址的设备、并连接

如果你确定周围有已知地址的蓝牙设备，或只需要连接指定地址的蓝牙设备，而忽略其他地址的设备，

```
bleManager.scanMacAndConnect(
        DEVICE_MAC,
        TIME_OUT,
        false,
        new BleGattCallback() {
            public void onNotFoundDevice() {
                Log.i(TAG, " 未发现设备！ ");
            }
            @Override
            public void onFoundDevice(BluetoothDevice device) {
                Log.i(TAG, " 发现设备 ： " + device.getAddress());
            }
            @Override
            public void onConnectSuccess(BluetoothGatt gatt, int status) {
                Log.i(TAG, " 连接成功！ ");
                gatt.discoverServices();
            }
            @Override
            public void onServicesDiscovered(BluetoothGatt gatt, int status) {
                Log.i(TAG, " 服务被发现！ ");
                bleManager.getBluetoothState();
            }
```

```
        @Override
        public void onConnectFailure(BleException exception) {
            Log.i(TAG, " 连接失败或连接中断:  " + exception.toString());
            bleManager.handleException(exception);
        }
    });
```

Notify,listen data changes through callback

参数中的 callback 和 uuid 将会形成关联，一旦此设备的 uuid 对应的 character 发生生数据变化，此 callback 将会回调此结果。此 callbak 将会唯一存在，和 uuid 是一一对应的关系。

```
bleManager.notify(
        UUID_SERVICE,
        UUID_NOTIFY,
        new BleCharacterCallback() {
            @Override
            public void onSuccess(BluetoothGattCharacteristic characteristic)
{
                Log.d(TAG, "notify result :  "
                        + String.valueOf(HexUtil.encodeHex(characteristic.getV
alue())));
            }
            @Override
            public void onFailure(BleException exception) {
                bleManager.handleException(exception);
            }
        });
```

stop notify,remove callback

```
bleManager.stopNotify(UUID_SERVICE, UUID_NOTIFY);
```

indicate,listen data changes through callback

```
bleManager.indicate(
        UUID_SERVICE,
        UUID_INDICATE,
```

```
        new BleCharacterCallback() {
            @Override
            public void onSuccess(BluetoothGattCharacteristic characteristic)
{
                Log.d(TAG, "indicate result :  "
                        + String.valueOf(HexUtil.encodeHex(characteristic.getV
alue())));
            }
            @Override
            public void onFailure(BleException exception) {
                Log.e(TAG, "indicate: " + exception.toString());
                bleManager.handleException(exception);
            }
        });
```

stop indicate,remove callback

```
bleManager.stopIndicate(UUID_SERVICE, UUID_INDICATE);
```

write

```
bleManager.writeDevice(
        UUID_SERVICE,
        UUID_WRITE,
        HexUtil.hexStringToBytes(SAMPLE_WRITE_DATA),
        new BleCharacterCallback() {
            @Override
            public void onSuccess(BluetoothGattCharacteristic characteristic)
{
                Log.d(TAG, "write result: "
                        + String.valueOf(HexUtil.encodeHex(characteristic.getV
alue())));
            }
            @Override
            public void onFailure(BleException exception) {
                Log.e(TAG, "write: " + exception.toString());
                bleManager.handleException(exception);
            }
        });
```

read

```
bleManager.readDevice(
        UUID_SERVICE,
        UUID_WRITE,
        new BleCharacterCallback() {
            @Override
            public void onSuccess(BluetoothGattCharacteristic characteristic)
{
                Log.d(TAG, "read result: "
                        + String.valueOf(HexUtil.encodeHex(characteristic.getV
alue())));
            }
            @Override
            public void onFailure(BleException exception) {
                Log.e(TAG, "read: " + exception.toString());
                bleManager.handleException(exception);
            }
        });
```

manual remove callback

uuid 作为参数，即不再监听这个 uuid 对应的 character 的数据变化，适用于移除 notify,indicate,write,read 对应的 callback 。

```
bleManager.stopListenCharacterCallback(UUID_NOTIFY);
```

复位（断开此次蓝牙连接，移除所有回调）

```
bleManager.closeBluetoothGatt();
```

其他

其他蓝牙操作可参考示例代码，或从 BleManager 这个类中开放的方法中找到。