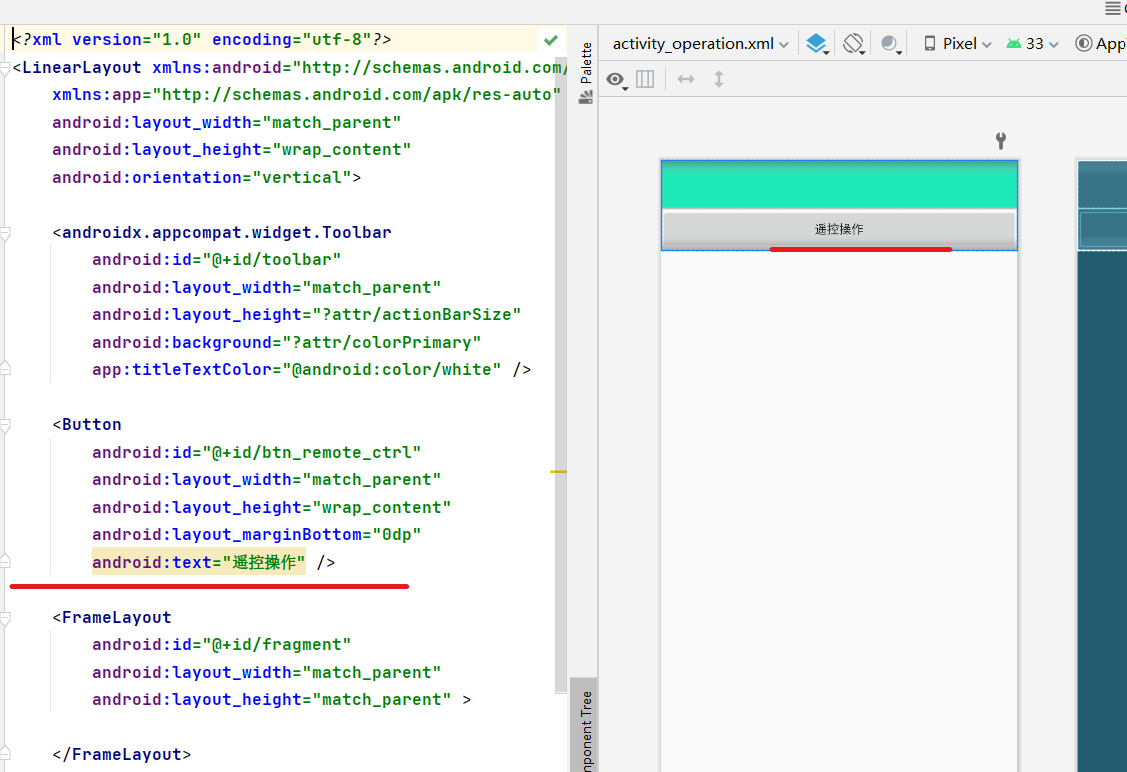
蓝牙demo地址：

<https://github.com/Jasonchenlijian/FastBle>

1 向自己的activity传递数据

1如下：在activity\_operation.xml UI布局中添加一个《跳转到自己控制界面》的按钮。activity\_ operation.xml 为软件的 《蓝牙操作activity》



2在OperationActivity 为刚刚的按钮 添加 《点击》监听事件，以跳转到自己的遥控 activity。其中remote\_ctr\_activity为自己的activity。KEY\_DATA, bleDevice为向遥控acitivity传递的蓝牙设备数据

**btn\_remote\_ctrl** = (Button)findViewById(R.id.***btn\_remote\_ctrl***);

**btn\_remote\_ctrl**.setOnClickListener(**new** View.OnClickListener() {  
 @Override  
 **public void** onClick(View view) {  
 **if** (BleManager.*getInstance*().isConnected(**bleDevice**)) {  
 Intent intent = **new** Intent(OperationActivity.**this**, remote\_ctr\_activity.**class**);  
 intent.putExtra(remote\_ctr\_activity.***KEY\_DATA***, **bleDevice**);  
 startActivity(intent);  
 }  
 }  
});

2 创建自己的遥控activity和UI

要点一：创建activity和UI

将自己的acitivity添加到AndroidManifest.xml的<**application**

下

<**activity android:name=".MyActivity.remote\_ctr\_activity"** />

在OperationActivity中添加点击按钮，跳转activity，将蓝牙设备信息传递给下一个activity

btn\_remote\_ctrl.setOnClickListener(**new** View.OnClickListener() {  
 @Override  
 **public void** onClick(View view) {  
 **if** (BleManager.*getInstance*().isConnected(**bleDevice**)) {  
 Intent intent = **new** Intent(OperationActivity.**this**, remote\_ctr\_activity.**class**);  
 intent.putExtra(remote\_ctr\_activity.***KEY\_DATA***, **bleDevice**);  
 startActivity(intent);  
 }  
 }  
});

要点二：获取蓝牙设备

在activity create时 获取上一个activity传递的蓝牙设备数据

**bleDevice** = getIntent().getParcelableExtra(***KEY\_DATA***); *//获取蓝牙设备***if** (**bleDevice** == **null**)  
 finish();

要点三：获取服务和特征

判断服务和特征的UUID是否为自己的ESP32设备：

**m\_service** = **null**;  
BluetoothGatt gatt = BleManager.*getInstance*().getBluetoothGatt(**bleDevice**);*//获取蓝牙服务***for** (BluetoothGattService service : gatt.getServices()) { *//查找指定服务* String uuid = service.getUuid().toString();  
 Log.*v*(**"aaa"**,**"========="**+uuid);  
 **if** (uuid.equals(**ESP32\_SERVICE\_UUID**)){ *//ESP32的指定服务* **m\_service** = service;  
 **break**;  
 }  
}  
*//查找指定特征***m\_characteristic** = **null**;  
**is\_m\_character** = **false**;  
**for** (BluetoothGattCharacteristic characteristic : **m\_service**.getCharacteristics()) {  
 String uuid = characteristic.getUuid().toString();  
 **if** (uuid.equals(**ESP32\_CHARACTER\_UUID**)){  
 **m\_characteristic** = characteristic;  
 **is\_m\_character** = **true**;  
 **break**;  
 }  
}

要点四：通过控件向蓝牙发送指令

为自己的按钮或其他控件绑定监听事件，如点击、按压松手等

*//前进按钮*Button btn\_front = findViewById(R.id.***btn\_move\_front***);  
btn\_front.setOnTouchListener(**new** View.OnTouchListener() {  
 @Override  
 **public boolean** onTouch(View view, MotionEvent motionEvent) {  
 **if** (motionEvent.getAction() == MotionEvent.***ACTION\_DOWN***){  
 *//发送按下指令* String hex = **"2C2C2C"**;  
 BleManager.*getInstance*().write(  
 **bleDevice**,  
 **ESP32\_SERVICE\_UUID**,  
 **ESP32\_CHARACTER\_UUID**,  
 HexUtil.*hexStringToBytes*(hex),  
 **m\_BleWriteCallback**);  
 }  
  
 **if** (motionEvent.getAction() == MotionEvent.***ACTION\_UP***) {  
 *//发送松手指令* String hex = **"2D2D2D"**;  
 BleManager.*getInstance*().write(  
 **bleDevice**,  
 **ESP32\_SERVICE\_UUID**,  
 **ESP32\_CHARACTER\_UUID**,  
 HexUtil.*hexStringToBytes*(hex),  
 **m\_BleWriteCallback** );  
 }  
 **return true**;  
 }  
});

3 接收下位机数据

要点一 打开通知

通过之前的步骤，重新建一个activity

在进入数据显示的activity后，开启蓝牙通知，实时接收下位机数据。因为下位机是通过 notify 实时发送数据的。

*//开启下位机通知，接收数据* BleManager.*getInstance*().notify(  
 **bleDevice**,  
 **ESP32\_SERVICE\_UUID**,  
 **ESP32\_CHARACTER\_UUID**,  
 **new** BleNotifyCallback() {  
  
 @Override  
 **public void** onNotifySuccess() {  
  
 runOnUiThread(**new** Runnable() {  
 @Override  
 **public void** run() {  
 *//Toast.makeText(data\_show\_activity.this,"打开实时数据成功",Toast.LENGTH\_LONG).show();* }  
 });  
 }  
  
 @Override  
 **public void** onNotifyFailure(**final** BleException exception) {  
 runOnUiThread(**new** Runnable() {  
 @Override  
 **public void** run() {  
*// Toast.makeText(data\_show\_activity.this,"打开实时数据失败:"+exception.toString(),Toast.LENGTH\_LONG).show();  
// Log.v("aa",exception.toString());* }  
 });  
 }  
 @Override  
 **public void** onCharacteristicChanged(**byte**[] data) {  
 runOnUiThread(**new** Runnable() {  
 @Override  
 **public void** run() {  
 *//****todo 解析下位机 实时数据（通知数据）*** *//addText(txt, HexUtil.formatHexString(characteristic.getValue(), true));* **byte** [] ESP32\_data = **m\_characteristic**.getValue();  
  
 **for** (**int** i = 0;i<ESP32\_data.**length**;i++){  
 ESP32\_Data\_Receive\_Prepare(ESP32\_data[i]);  
 }  
 }  
 });  
 }  
 });

其中ESP32\_Data\_Receive\_Prepare(ESP32\_data[i]); 是数据帧解码函数

要点二发送控制命令

*//开关实时数据按钮*Button btn\_get\_real\_data = findViewById(R.id.***btn\_get\_real\_time\_data***);  
btn\_get\_real\_data.setText(**"打开实时数据"**);  
btn\_get\_real\_data.setOnClickListener(**new** View.OnClickListener() {  
 @Override  
 **public void** onClick(View view) {  
  
 **if** (btn\_get\_real\_data.getText().toString().equals(**"打开实时数据"**)) {  
 btn\_get\_real\_data.setText(**"关闭实时数据"**);  
 *//发送按下指令* String hex = **"FF0A010102010000"**; *//打开实时数据* BleManager.*getInstance*().write(  
 **bleDevice**,  
 **ESP32\_SERVICE\_UUID**,  
 **ESP32\_CHARACTER\_UUID**,  
 HexUtil.*hexStringToBytes*(hex),  
 **m\_BleWriteCallback**);  
  
 } **else** {  
 btn\_get\_real\_data.setText(**"打开实时数据"**);  
 String hex = **"FF0A010102020000"**; *//关闭实时数据* BleManager.*getInstance*().write(  
 **bleDevice**,  
 **ESP32\_SERVICE\_UUID**,  
 **ESP32\_CHARACTER\_UUID**,  
 HexUtil.*hexStringToBytes*(hex),  
 **m\_BleWriteCallback**);  
 }  
 }  
});  
  
*//更新一次温度数据按钮*Button btn\_get\_a\_data = findViewById(R.id.***btn\_get\_a\_data***);  
btn\_get\_a\_data.setOnClickListener(**new** View.OnClickListener() {  
 @Override  
 **public void** onClick(View view) {  
 *//发送 获取温度数据 指令* String hex = **"FF0A020102010000"**;  
 BleManager.*getInstance*().write(  
 **bleDevice**,  
 **ESP32\_SERVICE\_UUID**,  
 **ESP32\_CHARACTER\_UUID**,  
 HexUtil.*hexStringToBytes*(hex),  
 **m\_BleWriteCallback**);  
  
 }  
});

下位机

向上位机发送数据的代码上位机的代码一致

所有数据帧参考dabble的数据帧

#define SERVICE\_UUID "e41ea377-7671-41aa-bc1a-abc4c38c8106" // 服务 UUID

#define CHARACTER\_UUID "cba1d466-344c-4be3-ab3f-189f80dd7518"//特征UUID

数据帧

# 帧含义

BYTE[0]:起始帧

Byte[1]:模式帧

Byte[2]:功能帧

Byte[3]:表示数据长度的所需要的字节，如Byte[3]=1 表示数据长度最长为255

Byte[4]:数据长度信息，如果Byte[3]=2,则数据长度信息占两位

Byte[5,+Byte[4]] 所有数据

Byte[end] 帧结束 00

# 下位机 to 上位机

## 实时数据

FF 01 01 01 04 00 00 00 00 00 定时器数据

## 非实时数据

FF 02 01 01 02 00 00 00 温度数据

# 上位机 to 下位机

FF 01 0 01 02 04 00 00 三角形前进 按压

FF 01 01 01 02 10 00 00 交叉后退 按压

FF 01 01 01 02 00 00 00 松手按钮

FF 0A 01 01 02 01 00 00 打开实时数据

FF 0A 01 01 02 02 00 00 关闭实时数据

FF 0A 01 01 02 03 00 00 获取单个实时数据一次《暂未使用》

FF 0A 02 01 02 01 00 00 获取温度数据一次