清華大學 Tsinghua University

蓝牙编程

嵌入式系统实践

0. 准备工作

- 安卓虚拟机AVD不支持蓝牙，
- 需要采用真实的安卓设备（手机、平板、树莓派）
- 推荐使用非鸿蒙系统（Harmony OS）的安卓手机

- 先在手机上使用蓝牙串口调试助手APP与单片机进行联合调试

1. 蓝牙权限申请

- 为了使APP能顺利使用蓝牙功能，应该设置相应的权限。蓝牙连接和通讯需要获取相关的蓝牙权限BLUETOOTH和BLUETOOTH_ADMIN。蓝牙权限是normal级权限，只需要在Manifest.xml里面声明即可，不需要判断和处理。

- Android6.0搜索周围的蓝牙设备，需要位置权限ACCESS_COARSE_LOCATION和ACCESS_FINE_LOCATION 其中的一个，并且将手机的位置服务（定位 GPS）打开。由于位置权限是dangerous级权限，除了需要在Manifest.xml里申请之外，还需要在代码中进行动态申请。
  - ACCESS_COARSE_LOCATION通过WiFi或移动基站获取粗略定位（误差在30~1500米)
  - ACCESS_FINE_LOCATION为GPS精确定位（精度10米以内）

```
<!--蓝牙有关权限-->
<uses-permission android:name="android.permission.BLUETOOTH"/>
<uses-permission android:name="android.permission.BLUETOOTH_ADMIN"/>
<!--定位有关权限-->
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
```

1. 蓝牙权限申请

自定义函数，查询定位是否打开

```
// 判断位置信息是否开启
private static boolean isLocationOpen(final Context context) {
    LocationManager manager = (LocationManager) context.getSystemService(Context.LOCATION_SERVICE);
    //gps定位
    boolean isGpsProvider = manager.isProviderEnabled(LocationManager.GPS_PROVIDER);
    //网络定位
    boolean isNetWorkProvider = manager.isProviderEnabled(LocationManager.NETWORK_PROVIDER);
    return isGpsProvider || isNetWorkProvider;
}
```

动态申请权限 （在OnCreate函数中）

```
if (Build.VERSION.SDK_INT >= 23) {
    boolean isLocat = isLocationOpen(getApplicationContext());
    Toast.makeText(getApplicationContext(), "isLo:" + isLocat, Toast.LENGTH_SHORT).show();
    //开启位置服务，支持获取ble蓝牙扫描结果
    if (!isLocat) {
        Intent enableLocate = new Intent(Settings.ACTION_LOCATION_SOURCE_SETTINGS);
        startActivityForResult(enableLocate, 1);
    }
}
```

2. 蓝牙编程的几个重要类

- **BluetoothAdapter类**

本地的蓝牙适配器。是所有蓝牙交互操作的入口点。通过这个类可以发现其他蓝牙设备，查询已配对的设备列表。我们使用一个已知的MAC地址来实例化一个BluetoothDevice，以及创建一个BluetoothServerSocket来为监听与其他设备的通信。

- **BluetoothDevice类**

远程蓝牙设备。使用这个类来请求一个与远程设备的BluetoothSocket连接，或者查询关于设备名称、地址、类和连接状态等设备信息。

- **BluetoothSocket类**

代表一个蓝牙Socket接口（和TCP Socket类似），它允许一个应用与其他蓝牙设备通过InputStream和OutputStream交换数据。

- **BluetoothServerSocket类**

代表一个开放的服务器Socket，它监听接受的请求（与TCP ServerSocket类似）。为了连接两台Android设备，一个设备必须使用这个类开启一个服务器socket。当一个远程蓝牙设备开始一个和该设备的连接请求，BluetoothServerSocket将会返回一个已连接的BluetoothSocket，接受该连接。

更多信息：https://blog.csdn.net/weixin_56291477/article/details/123413036

3. 蓝牙编程的步骤

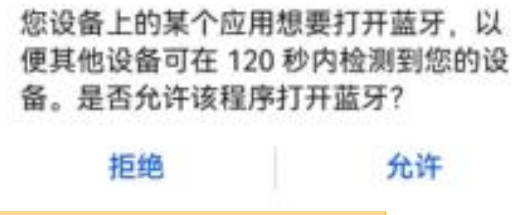
Step1：在APP端首先实例化一个BluetoothAdapter类的对象

```
BluetoothAdapter  bluetoothAdapter = BluetoothAdapter.getDefaultAdapter();
```

如果设备不支持蓝牙，则返回null

Step 2：打开蓝牙。调用适配器对象的isEnable方法判断蓝牙是否已经打开。如果没有打开，则通过调用适配器对象的enable方法打开蓝牙。

Step 3：允许蓝牙可见。通过Intent的方式向用户请求允许蓝牙被发现




```
if (!bluetoothAdapter.isEnabled())
{
    bluetoothAdapter.enable();//开启蓝牙
    Intent enable = new Intent(BluetoothAdapter.ACTION_REQUEST_DISCOVERABLE);
    //120秒为蓝牙设备可见时间
    enable.putExtra(BluetoothAdapter.EXTRA_DISCOVERABLE_DURATION, 120);
    startActivity(enable);
}
```

更多信息：https://blog.csdn.net/weixin_56291477/article/details/123413036

3. 蓝牙编程的步骤

Step 4: 搜索蓝牙设备

```
if (!bluetoothAdapter.isEnabled())
{
    Toast.makeText(getApplicationContext(), "请先开启蓝牙", Toast.LENGTH_SHORT).show();
}
else
{
    bluetoothAdapter.startDiscovery();
    Toast.makeText(getApplicationContext(), "开始搜索", Toast.LENGTH_SHORT).show();
    IntentFilter filter = new IntentFilter(BluetoothDevice.ACTION_FOUND);
    registerReceiver(receiver, filter); //注册广播接收器
}
```

IntentFilter，应用程序自己告诉Android系统自己能响应、处理哪些隐式Intent

```
//蓝牙广播处理
receiver = new BroadcastReceiver() {
    public void onReceive(Context context, Intent intent) {
        String action = intent.getAction();
        //搜索设备时，取得设备的MAC地址
        if (BluetoothDevice.ACTION_FOUND.equals(action))
        {
            BluetoothDevice device = intent.getParcelableExtra(BluetoothDevice.EXTRA_DEVICE);
            String str = device.getName() + "|" + device.getAddress();
        }
    }
};
```

3. 蓝牙编程的步骤

Step 5: 连接蓝牙设备

```
Method clientMethod = device.getClass().getMethod("createRfcommSocket", new Class[]{int.class});
mSocket = (BluetoothSocket) clientMethod.invoke(device, 3);
try
{
    mSocket.connect();//连接
    textStatus.setText("尝试连接.....");
    if (mSocket.isConnected())
    {
        textStatus.setText("连接成功");
        Toast.makeText(getApplicationContext(), "蓝牙连接成功", Toast.LENGTH_SHORT).show();
        mThread= new ComThread(mSocket);
        mThread.start();//另开一个线程，与蓝牙设备进行通信
        state = CONNECTED;
    }
    else
    {
        textStatus.setText("连接失败");
        Toast.makeText(getApplicationContext(), "蓝牙连接失败", Toast.LENGTH_SHORT).show();
        mSocket.close();
        listView.setVisibility(View.VISIBLE);
    }
} catch (Exception e) {
    e.printStackTrace();
}
```

一般为1，鸿蒙系统用3

3. 蓝牙编程的步骤

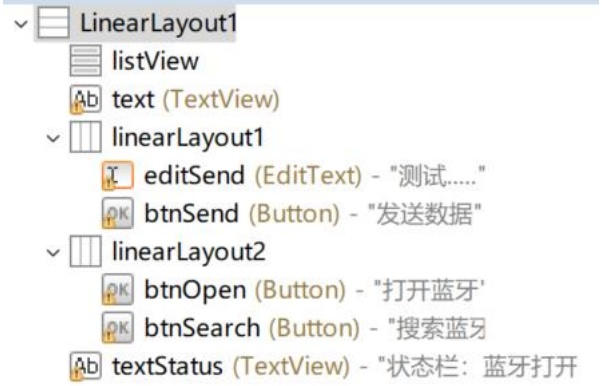
通信线程

```
class ComThread extends Thread{
    private BluetoothSocket s;
    private boolean exitflag = false;
    public ComThread(BluetoothSocket s) {this.s=s;}
    public synchronized boolean getFlag(){return exitflag;}
    public synchronized void  setFlag(boolean v) {exitflag=true;}

    private InputStream inputStream;
    private OutputStream outputStream;

    public void write(byte[] bytes)
    {
        try
        {
            outputStream = mSocket.getOutputStream();
            outputStream.write(bytes);
        }
        catch (IOException e)
        {
            e.printStackTrace();
        }
    }
```

```
public void run() {
    try
    {
        inputStream=s.getInputStream();
        byte[] data = new byte[1024];
        int len = 0;
        String result = "";

        while (!exitflag)
        {
            if (inputStream.available() <= 0)
                continue;
            else
            {
                try
                {
                    Thread.sleep(500);//等待0.5秒，让数据接收完整
                    len = inputStream.read(data);
                    result = URLDecoder.decode(new String(data, "utf-8"));

                    final String mid = result;
                    textData.post(new Runnable() {
                        public void run() {
                           textData.append(mid+"\r\n");
                        }
                    });

                }catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
        }
        inputStream.close();
        outputStream.close();
        s.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

使用View.post(Runnable r)更新界面中的TextView

案例



- ListView 用来显示搜索到的蓝牙设备
  - 初始为空，搜索之后添加找到的蓝牙设备
  - 单击某个Item会触发事件，在事件响应函数中进行配对和Socket连接
  - Socket连接成功后，ListView会被隐藏
- TextView 用来显示发送/接收的消息
  - 初始时隐藏，Socket连接成功后显示
- EditText 用来输入要发送的消息

一些注意事项

- 搜索周边蓝牙设备时，本机并不需要处于蓝牙可见状态，但其他设备必须处于蓝牙可见状态，本机才可以搜索到。

- 搜索周边设备对于蓝牙适配器而言是一个非常繁重的操作过程，并且会消耗大量资源。在找到要连接的设备后，确保始终使用 cancelDiscovery() 停止发现，然后再尝试连接。此外，如果已经保持与某台设备的连接，那么执行搜索操作可能会大幅减少可用于该连接的带宽，因此不应该在处于连接状态时执行搜索周边蓝牙的操作。

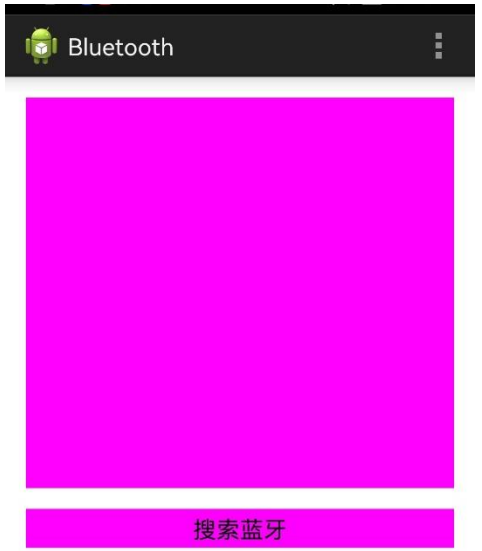
- 在调用 connect() 时，应始终确保设备未在进行搜索操作。如果正在进行搜索操作，则会大幅降低连接尝试的速度，并增加连接失败的可能性。

任务1：搜索蓝牙设备

- 实现一个简单Android界面，包含1个listView和1个Button。
- 要求：
  - 按下Button按钮时，若蓝牙已打开，弹出“开始搜索”，并在listView中显示周围的蓝牙设备名称和MAC地址，中间用“|”分开；
  - 按下Button按钮时，若蓝牙未打开，弹出“请先开启蓝牙”。
- 提示：
  - 调用BluetoothAdapter类中的startDiscovery()方法可以搜索附近的蓝牙设备；
  - 调用BluetoothDevice.getAddress()获取MAC地址；
  - 搜索到附近蓝牙设备的时候，系统会发出广播BluetoothDevice.ACTION_FOUND。

任务2：连接蓝牙设备

- 在前一界面的基础上，继续实现新功能。
- 要求：
  - 按下Item时，若蓝牙已打开，则连接相应设备，若连接成功，将listView替换为1个Textview；
  - 按下Item时，若蓝牙未打开，则弹出“请先开启蓝牙”。
- 提示：
  - 构造onItemClick()方法实现对listView的点击操作（类似于OnClick，参数不同）；
  - 调用setVisibility()方法更改图形模块的可见性（View.GONE，View.VISIBLE）；
  - 调用BluetoothSocket.connect()方法连接蓝牙。



任务2：相关代码

注册监听器

```
listView.setAdapter(deviceAdapter);
listView.setOnItemClickListener(this);
```

设置textDate对象

```
textData=(TextView) this.findViewById(R.id.text);
```

```
private TextView textData;
public  BluetoothSocket mSocket;

public void onItemClick(AdapterView<?> adapterView, View view, int position, long l) {

    if (!bluetoothAdapter.isEnabled())
    {
        Toast.makeText(getApplicationContext(), "请先开启蓝牙", Toast.LENGTH_SHORT).show();
    }
    else
    {
        bluetoothAdapter.cancelDiscovery();//停止搜索

        String str = listDevices.get(position);
        String macAdress = str.split("\\|")[1];

        BluetoothDevice device = bluetoothAdapter.getRemoteDevice(macAdress);
        try
        {
```

任务2：相关代码

```
            */


            Method clientMethod = device.getClass().getMethod("createRfcommSocket", new Class[]{int.class});
            mSocket = (BluetoothSocket) clientMethod.invoke(device, 1);
            // 1 连接单片机 ,  2和3 连接手机


            try
            {
                mSocket.connect();//连接
                if (mSocket.isConnected())
                {
                    Toast.makeText(getApplicationContext(), "蓝牙连接成功", Toast.LENGTH_SHORT).show();
                    listView.setVisibility(View.GONE);
                    textData.setVisibility(View.VISIBLE);
                }
                else
                {
                    Toast.makeText(getApplicationContext(), "蓝牙连接失败", Toast.LENGTH_SHORT).show();
                    mSocket.close();
                    listView.setVisibility(View.VISIBLE);
                }
            } catch (Exception e) {
                e.printStackTrace();
            }

        } catch (NoSuchMethodException e) {
            e.printStackTrace();
        } catch (IllegalAccessException e) {
            e.printStackTrace();
        } catch (InvocationTargetException e) {
            e.printStackTrace();
        }
    }
}
```

任务3：蓝牙设备相互通讯

- 在前一界面的基础上，继续实现新功能。
- 要求：
  - 添加一个EditText和一个Button，如右图；
  - 连接蓝牙设备后，在EditText中输入消息，按下Button实现蓝牙数据发送；
  - 连接蓝牙设备后，在串口调试助手中输入并发送消息，在手机端实现蓝牙数据接收，显示在新出现的TextView中。
- 提示：
  - 采用多线程方法，蓝牙通讯在另一个线程中，防止主程序的阻塞；
  - 调用Thread.sleep()使数据接收完整。

任务3：相关代码

```
private final int CONNECTED = 1000;
private final int DISCONNECTED = 1001;
private int state = DISCONNECTED;
```

```
private ComThread mThread = null;
//private ConnectThread mThread = null;
```

```
case R.id.btnSend:
    if (state == CONNECTED && mThread != null)
    {
        String info = editSend.getText().toString();

        textStatus.setText("Send:"+ info);
        textData.setText("Send Data:"+info+"\r\n");
        //textData.append("Send Data:"+info+"\r\n");
        mThread.write(info.getBytes());
    }
    break;
```

任务3：相关代码

```
class ComThread extends Thread{

    private BluetoothSocket s;
    private boolean exitflag = false;
    public ComThread(BluetoothSocket s) {this.s=s;}
    public synchronized boolean getFlag(){return exitflag;}
    public synchronized void  setFlag(boolean v) {exitflag=true;}

    private InputStream inputStream;
    private OutputStream outputStream;

    public void write(byte[] bytes)
    {
        try
        {
            outputStream = mSocket.getOutputStream();
            outputStream.write(bytes);
        }
        catch (IOException e)
        {
            e.printStackTrace();
        }
    }
```

任务3：相关代码

```
@Override
public void run() {
    try
    {
        inputStream=s.getInputStream();

        int len = 0;
        String result = "";

        exitflag=false;
        while (len != -1)
        {
            if (inputStream.available() <= 0)
            {
                Thread.sleep(1000);//等待0.5秒，让数据接收完整
                continue;
            }
            else
            {
                try
                {
                    Thread.sleep(500);//等待0.5秒，让数据接收完整
                    byte[] data = new byte[1024];
                    len = inputStream.read(data);
                    result = URLDecoder.decode(new String(data, "utf-8"));
```

任务3：相关代码

```
                    final String mid = result;
                    textData.post(new Runnable() {
                        public void run() {
                            //textData.append("Receiv: "+mid);
                            textData.setText("Receiv: "+mid);
                            //String oldstr = textData.getText().toString();
                            //textData.setText(oldstr+"\r\nReceiv: "+mid);
                        }
                    });
                }catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
        }
        inputStream.close();
        outputStream.close();
        s.close();
    } catch (IOException e) {
        e.printStackTrace();
    } catch (InterruptedException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
  }
}
```