Ophone 平台蓝牙编程之蓝牙聊天分析（一）

OPhone 平台开发, 2010-10-25 16:39:52

标签 ：OPhone 蓝牙 编程

上一篇文章我们分析了 Ophone 平台的蓝牙开发包，本文我们将通过学习 android 的蓝牙聊天示例应用程序来介绍蓝牙开发包的使用，该示例程序完整的包含了蓝牙开发的各个部分，将实现两个设备通过蓝牙进行连接并聊天。

AndroidManifest.xml

前面我们说过，在使用蓝牙 API 时就需要开启某些权限，同时我们还可以从 AndroidManifest.xml 文件中找到应用程序启动时所进入的界面 Activity 等信息，因此下面我们首先打开 AndroidManifest.xml 文件，代码如下：

view plaincopy to clipboardprint?

```
1. <manifest xmlns:android="http://schemas.android.com/apk/res/android"

2.     package="com.example.android.BluetoothChat"
3.     android:versionCode="1"
4.     android:versionName="1.0">
5.     <!-- 最小的 sdk 版本 -->>
6.     <uses-sdk minSdkVersion="6" />
7.     <!-- 权限申明 -->>
8.
   <uses-permission android:name="android.permission.BLUETOOTH_ADMIN
   " />
9.
   <uses-permission android:name="android.permission.BLUETOOTH" />
10.
11.    <application android:label="@string/app_name"
12.    android:icon="@drawable/app_icon" >
13.    <!-- 默认 Activity -->>
14.    <activity android:name=".BluetoothChat"
15.    android:label="@string/app_name"
16.    android:configChanges="orientation|keyboardHidden">
17.    <intent-filter>
18.    <action android:name="android.intent.action.MAIN" />
19.    <category android:name="android.intent.category.LAUNCHER" />
20.    </intent-filter>
21.    </activity>
22.    <!-- 用于显示蓝牙设备列表的 Activity -->
23.    <activity android:name=".DeviceListActivity"
```

```
24.     android:label="@string/select_device"
25.     android:theme="@android:style/Theme.Dialog"
26.     android:configChanges="orientation|keyboardHidden" />
27.     </application>
28.     </manifest>
```

首先 minSdkVersion 用于说明该应用程序所需要使用的最小 SDK 版本，这里设置为 6，也就是说最小需要使用 android1.6 版本的 sdk，同时 Ophone 则需要使用 oms2.0 版本，然后打开了 BLUETOOTH 和 BLUETOOTH_ADMIN 两个蓝牙操作相关的权限，最后看到了两个 Activity 的声明，他们分别是 BluetoothChat（默认主 Activity）和 DeviceListActivity（显示设备列表），其中 DeviceListActivity 风格被定义为一个对话框风格，下面我们将分析该程序的每个细节。

BluetoothChat

首先，程序启动进入 BluetoothChat，在 onCreate 函数中对窗口进行了设置，代码如下：

view plaincopy to clipboardprint?

```
1.  // 设置窗口布局
2.      requestWindowFeature(Window.FEATURE_CUSTOM_TITLE);
3.      setContentView(R.layout.main);
4.
    getWindow().setFeatureInt(Window.FEATURE_CUSTOM_TITLE, R.layout.custom_title);
```

这里可以看到将窗口风格设置为自定义风格了，并且指定了自定义 title 布局为 custom_title，其定义代码如下：

view plaincopy to clipboardprint?

```
1.  <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
2.      android:layout_width="match_parent"
3.      android:layout_height="match_parent"
4.      android:gravity="center_vertical"
5.      >
6.      <TextView android:id="@+id/title_left_text"
7.      android:layout_alignParentLeft="true"
8.      android:ellipsize="end"
9.      android:singleLine="true"
10.     style="?android:attr/windowTitleStyle"
11.     android:layout_width="wrap_content"
```

```
12.     android:layout_height="match_parent"
13.     android:layout_weight="1"
14.     />
15.     <TextView android:id="@+id/title_right_text"
16.     android:layout_alignParentRight="true"
17.     android:ellipsize="end"
18.     android:singleLine="true"
19.     android:layout_width="wrap_content"
20.     android:layout_height="match_parent"
21.     android:textColor="#fff"
22.     android:layout_weight="1"
23.     />
24.     </RelativeLayout>
```

该布局将 title 设置为一个相对布局 RelativeLayout，其中包含了两个 TextView，一个在左边一个在右边，分别用于显示应用程序的标题 title 和当前的蓝牙配对链接名称，如下图所示。

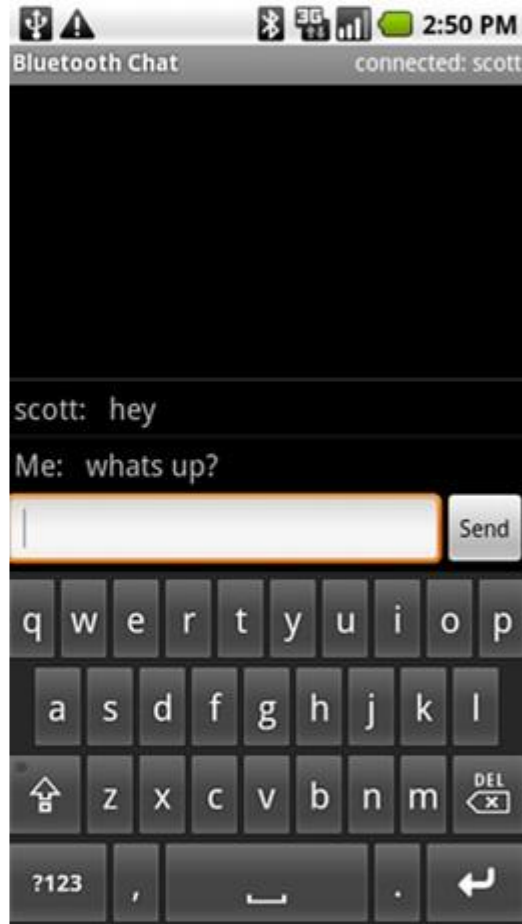
其中左边显示为应用程序名称"BluetoothChat",右边显示一个 connected：scort 则表示当前配对成功正在进行聊天的链接名称。整个聊天界面的布局在 main.xml 中实现，代码如下：

view plaincopy to clipboardprint?

```
1. <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
2.     android:orientation="vertical"
3.     android:layout_width="match_parent"
4.     android:layout_height="match_parent"
5.     >
6.     <!-- 显示设备列表 -->
7.     <ListView android:id="@+id/in"
8.     android:layout_width="match_parent"
9.     android:layout_height="match_parent"
10.    android:stackFromBottom="true"
11.    android:transcriptMode="alwaysScroll"
12.    android:layout_weight="1"
13.    />
14.    <!-- 显示发送消息的编辑框 -->
15.    <LinearLayout
16.    android:orientation="horizontal"
```

```
17.      android:layout_width="match_parent"
18.      android:layout_height="wrap_content"
19.      >
20.      <EditText android:id="@+id/edit_text_out"
21.      android:layout_width="wrap_content"
22.      android:layout_height="wrap_content"
23.      android:layout_weight="1"
24.      android:layout_gravity="bottom"
25.      />
26.      <Button android:id="@+id/button_send"
27.      android:layout_width="wrap_content"
28.      android:layout_height="wrap_content"
29.      android:text="@string/send"
30.      />
31.      </LinearLayout>
32.      </LinearLayout>
```

整个界面的布局将是一个线性布局 LinearLayout，其中包含了另一个 ListView（用于显示聊天的对话信息）和另外一个线性布局来实现一个发送信息的窗口，发送消息发送框有一个输入框和一个发送按钮构成。整个界面如下图所示。



布局好界面，下面我们需要进入编码状态，首先看 BluetoothChat 所要那些成员变量，如下代码所示：

view plaincopy to clipboardprint?

```
1.  public class BluetoothChat extends Activity {
2.          // Debugging
3.          private static final String TAG = "BluetoothChat";
4.          private static final boolean D = true;
5.
6.          //从 BluetoothChatService Handler 发送的消息类型
7.          public static final int MESSAGE_STATE_CHANGE = 1;
8.          public static final int MESSAGE_READ = 2;
9.          public static final int MESSAGE_WRITE = 3;
10.         public static final int MESSAGE_DEVICE_NAME = 4;
11.         public static final int MESSAGE_TOAST = 5;
12.
13.         // 从 BluetoothChatService Handler 接收消息时使用的键名(键-值模型)
14.         public static final String DEVICE_NAME = "device_name";
15.         public static final String TOAST = "toast";
```

```
16.
17.     // Intent 请求代码（请求链接，请求可见）
18.     private static final int REQUEST_CONNECT_DEVICE = 1;
19.     private static final int REQUEST_ENABLE_BT = 2;
20.
21.     // Layout Views
22.     private TextView mTitle;
23.     private ListView mConversationView;
24.     private EditText mOutEditText;
25.     private Button mSendButton;
26.
27.     // 链接的设备的名称
28.     private String mConnectedDeviceName = null;
29.     // Array adapter for the conversation thread
30.     private ArrayAdapter<String> mConversationArrayAdapter;
31.     // 将要发送出去的字符串
32.     private StringBuffer mOutStringBuffer;
33.     // 本地蓝牙适配器
34.     private BluetoothAdapter mBluetoothAdapter = null;
35.     // 聊天服务的对象
36.     private BluetoothChatService mChatService = null;
37.     //......
```

其中 Debugging 部分则将用于我们在调试程序时通过 log 打印日志用，其他部分我们都加入了注释，需要说明的是 BluetoothChatService ，它是我们自己定义的一个用来管理蓝牙的端口监听，链接，管理聊天的程序，后面我们会介绍。在这里需要说明一点，这些代码都出自 google 的员工之手，大家在学习时，可以借鉴很多代码编写的技巧和风格，这都将对我们有非常大的帮助。

然后，我们就需要对界面进行一些设置，如下代码将用来设置我们自定义的标题 title 需要显示的内容：

view plaincopy to clipboardprint?

```
1.  // 设置自定义 title 布局
2.      mTitle = (TextView) findViewById(R.id.title_left_text);
3.      mTitle.setText(R.string.app_name);
4.      mTitle = (TextView) findViewById(R.id.title_right_text);
```

左边的 TextView 被设置为显示应用程序名称，右边的则需要我们在链接之后在设置更新，目前则显示没有链接字样,所以这里我们暂不设置,进一步就需要获取本地蓝牙适配器

BluetoothAdapter 了，因为对于有关蓝牙的任何操作都需要首先获得该蓝牙适配器，获取代码非常简单，如下：

view plaincopy to clipboardprint?

```
1. // 得到一个本地蓝牙适配器
2.         mBluetoothAdapter = BluetoothAdapter.getDefaultAdapter();
3.
4.         // 如果适配器为 null，则不支持蓝牙
5.         if (mBluetoothAdapter == null) {
6.
   Toast.makeText(this, "Bluetooth is not available", Toast.LENGTH_LONG).s
   how();
7.         finish();
8.         return;
9.         }
10.        getDefaultAdapter()函数用于获取本地蓝牙适配器，然后检测是否为 null，如
   果为 null 则表示没有蓝牙设备的支持，将通过 toast 告知用户。
11.        在 onStart()函数中，我们将检测蓝牙是否被打开，如果没有打开，则请求打开，
   否则就可以设置一些聊天信息的准备工作，代码如下：
12.        @Override
13.        public void onStart() {
14.        super.onStart();
15.        if(D) Log.e(TAG, "++ ON START ++");
16.
17.        // 如果蓝牙没有打开，则请求打开
18.        // setupChat() will then be called during onActivityResult
19.        if (!mBluetoothAdapter.isEnabled()) {
20.
   Intent enableIntent = new Intent(BluetoothAdapter.ACTION_REQUEST_E
   NABLE);
21.        startActivityForResult(enableIntent, REQUEST_ENABLE_BT);
22.        // 否则，设置聊天会话
23.        } else {
24.        if (mChatService == null) setupChat();
25.        }
26.        }
```

如果蓝牙没有打开，我们则通过 BluetoothAdapter.ACTION_REQUEST_ENABLE 来请求打开蓝牙，REQUEST_ENABLE_BT 则是我们自己定义的用于请求打开蓝牙的 Intent 代码，最后当我们调用 startActivityForResult 来执行请求时，就会在 onActivityResult 函数中得到一个反

馈，如果当前蓝牙已经打开，那么就可以调用 setupChat 函数来准备蓝牙聊天相关的工作，稍后分析该函数的具体实现。

下面我们分析一下请求打开蓝牙之后，在 onActivityResult 中得到的反馈信息，我们传递了 REQUEST_ENABLE_BT 代码作为请求蓝牙打开的命令，因此在 onActivityResult 中，需要会得到一个请求代码为 REQUEST_ENABLE_B 的消息，对于其处理如下代码所示：

view plaincopy to clipboardprint?

```
1. case REQUEST_ENABLE_BT:
2.        // 在请求打开蓝牙时返回的代码
3.        if (resultCode == Activity.RESULT_OK) {
4.        // 蓝牙已经打开，所以设置一个聊天会话
5.        setupChat();
6.        } else {
7.        // 请求打开蓝牙出错
8.        Log.d(TAG, "BT not enabled");
9.
   Toast.makeText(this, R.string.bt_not_enabled_leaving, Toast.LENGTH_SH
   ORT).show();
10.       finish();
11.       }
```

在请求时，如果返回代码为 Activity.RESULT_OK，则表示请求打开蓝牙成功，那么我们就可以和上面的操作进度一样，调用 setupChat 来设置蓝牙聊天相关信息，如果返回其他代码，则表示请求打开蓝牙失败，这时我们同样通过一个 Toast 来告诉用户，同时也需要调用 finish() 函数来结束应用程序。

如果打开蓝牙无误，那么下面我们开始进入 setupChat 的设置，其代码实现如下：

view plaincopy to clipboardprint?

```
1. private void setupChat() {
2.        Log.d(TAG, "setupChat()");
3.
4.        // 初始化对话进程
5.
   mConversationArrayAdapter = new ArrayAdapter<String>(this, R.layout.
   message);
6.        // 初始化对话显示列表
7.        mConversationView = (ListView) findViewById(R.id.in);
8.        // 设置话显示列表源
9.        mConversationView.setAdapter(mConversationArrayAdapter);
10.
```

```
11.     // 初始化编辑框，并设置一个监听，用于处理按回车键发送消息
12.     mOutEditText = (EditText) findViewById(R.id.edit_text_out);
13.     mOutEditText.setOnEditorActionListener(mWriteListener);
14.
15.     // 初始化发送按钮，并设置事件监听
16.     mSendButton = (Button) findViewById(R.id.button_send);
17.     mSendButton.setOnClickListener(new OnClickListener() {
18.     public void onClick(View v) {
19.     // 取得 TextView 中的内容来发送消息
20.     TextView view = (TextView) findViewById(R.id.edit_text_out);
21.     String message = view.getText().toString();
22.     sendMessage(message);
23.     }
24.     });
25.
26.     // 初始化 BluetoothChatService 并执行蓝牙连接
27.     mChatService = new BluetoothChatService(this, mHandler);
28.
29.     // 初始化将要发出的消息的字符串
30.     mOutStringBuffer = new StringBuffer("");
31.     }
```

首先构建一个对话进程 mConversationArrayAdapter，然后从 xml 中取得用于显示对话信息的列表 mConversationView，最后将列表的数据来源 Adapter 设置为 mConversationArrayAdapter，这里我们可以看到 mConversationArrayAdapter 所指定的资源为 message.xml，其定义实现如下：

view plaincopy to clipboardprint?

```
1.  <TextView xmlns:android="http://schemas.android.com/apk/res/android"

2.      android:layout_width="match_parent"
3.      android:layout_height="wrap_content"
4.      android:textSize="18sp"
5.      android:padding="5dp"
6.      />
```

很简单，就包含了一个 TextView 用来显示对话内容即可，这里设置了文字标签的尺寸大小 textSize 和 padding 属性。

然后我们取得了编辑框 mOutEditText，用于输入聊天内容的输入框，并对其设置了一个事件监听 mWriteListener，其监听函数的实现如下：

view plaincopy to clipboardprint?

```
1. // The action listener for the EditText widget, to listen for the return key
2.     private TextView.OnEditorActionListener mWriteListener =
3.     new TextView.OnEditorActionListener() {
4.
   public boolean onEditorAction(TextView view, int actionId, KeyEvent eve
   nt) {
5.     // 按下回车键并且是按键弹起的事件时发送消息
6.
   if (actionId == EditorInfo.IME_NULL && event.getAction() == KeyEvent.A
   CTION_UP) {
7.     String message = view.getText().toString();
8.     sendMessage(message);
9.     }
10.    if(D) Log.i(TAG, "END onEditorAction");
11.    return true;
12.    }
13.    };
```

首先在其监听中实现了 onEditorAction 函数，我们通过判断其参数 actionId 来确定事件触发的动作，其中的"EditorInfo.IME_NULL"在 Ophone 中表示回车键消息，然后再加上 KeyEvent.ACTION_UP，则表示当用户按下回车键并弹起时才触发消息的处理，处理过程也很简单，将输入框中内容取出到变量 message 中，然后调用 sendMessage 函数来发送一条消息，具体的发送细节，我们稍后分析。

在 setupChat 函数中，我们还对发送消息的按钮进行的初始化，同样为其设置了事件监听（setOnClickListener），监听的内容则也是取得输入框中的信息，然后调用 sendMessage 函数来发送消息，和用户按回车键来发送消息一样。

最后一个重要的操作就是初始化了 BluetoothChatService 对象 mChatService 用来管理蓝牙的链接，聊天的操作,并且设置了其 Handler 对象 mHandler 来负责数据的交换和线程之间的通信。另外还准备了一个空的字符串对象 mOutStringBuffer，用于当我们在发送消息之后，对输入框的清理。

应用菜单

该应用程序除了这些界面的布局之外，我们还为其设置了一个菜单，菜单包括了"扫描设备"和"使设备可见（能够被其他设备所搜索到）"，创建菜单的方式有很多种，这里 gogole 的员工，

比较喜欢和推崇使用 xml 布局（将界面和逻辑分开），所以我们首先看一下对于该应用程序通过 xml 所定义的菜单布局，代码如下：

view plaincopy to clipboardprint?

```
1.  <menu xmlns:android="http://schemas.android.com/apk/res/android">
2.          <!-- 扫描菜单 -->
3.         <item android:id="@+id/scan"
4.         android:icon="@android:drawable/ic_menu_search"
5.         android:title="@string/connect" />
6.         <!-- 可见操作 -->
7.         <item android:id="@+id/discoverable"
8.         android:icon="@android:drawable/ic_menu_mylocation"
9.         android:title="@string/discoverable" />
10.        </menu>
```

这样的定义的确非常的清晰，我们可以随意向这个 Menu 中添加菜单选项（itme）,这里就定义了上面我们所说的两个菜单。然后再程序中通过 onCreateOptionsMenu 函数中来装载该菜单布局，遂于菜单的点击可以通过 onOptionsItemSelected 函数的不同参数来辨别，下面是该应用程序中对菜单选项的处理和装载菜单布局：

view plaincopy to clipboardprint?

```
1.  //创建一个菜单
2.         @Override
3.         public boolean onCreateOptionsMenu(Menu menu) {
4.         MenuInflater inflater = getMenuInflater();
5.         inflater.inflate(R.menu.option_menu, menu);
6.         return true;
7.         }
8.         //处理菜单事件
9.         @Override
10.        public boolean onOptionsItemSelected(MenuItem item) {
11.        switch (item.getItemId()) {
12.        case R.id.scan:
13.        // 启动 DeviceListActivity 查看设备并扫描
14.        Intent serverIntent = new Intent(this, DeviceListActivity.class);
15.        startActivityForResult(serverIntent, REQUEST_CONNECT_DEVICE);
16.        return true;
17.        case R.id.discoverable:
18.        // 确保设备处于可见状态
19.        ensureDiscoverable();
```

```
20.     return true;
21.     }
22.     return false;
23.     }
```

装载菜单布局的时候使用了 MenuInflater 对象，整个过程很简单，大家可以参考上面的代码实现，在处理菜单事件的时候，通过 item.getItemId()我们可以得到当前选择的菜单项的 ID，首先是扫描设备（R.id.scan），这里我们有启动了另外一个 Activity 来专门处理扫描设备 DeviceListActivity，如果扫描之后我们将通过 startActivityForResult 函数来请求链接该设备，同样我们也会在 onActivityResult 函数中收到一个反馈信息，命令代码为 REQUEST_CONNECT_DEVICE，如果反馈的请求代码为 Activity.RESULT_OK，则表示扫描成功（扫描过程我们稍后介绍），那么下面就可以开始准备链接了，实现代码如下：

view plaincopy to clipboardprint?

```
1. case REQUEST_CONNECT_DEVICE:
2.      // 当 DeviceListActivity 返回设备连接
3.      if (resultCode == Activity.RESULT_OK) {
4.      // 从 Intent 中得到设备的 MAC 地址
5.      String address = data.getExtras()
6.      .getString(DeviceListActivity.EXTRA_DEVICE_ADDRESS);
7.      // 得到蓝牙设备对象
8.
   BluetoothDevice device = mBluetoothAdapter.getRemoteDevice(address);

9.      // 尝试连接这个设备
10.     mChatService.connect(device);
11.     }
12.     break;
```

首先我们可以通过 DeviceListActivity.EXTRA_DEVICE_ADDRESS 来取得设备的 Mac 地址，然后通过 Mac 地址使用蓝牙适配器 mBluetoothAdapter 的 getRemoteDevice 函数来查找到该地址的设备 BluetoothDevice，查询到之后我们可以通过 mChatService 对象的 connect 来链接该设备。

上面我们说的是扫描蓝牙设备并链接的过程，一般蓝牙设备在打开之后都需要设置可见状态，下面我们来看一下另一个菜单选项的实现，用于使设备处于可见状态，其菜单项的 ID 为 R.id.discoverable，具体实现过程则位于 ensureDiscoverable 函数中,其实现如下代码：

view plaincopy to clipboardprint?

```
1. private void ensureDiscoverable() {
2.       if(D) Log.d(TAG, "ensure discoverable");
3.       //判断扫描模式是否为既可被发现又可以被连接
4.       if (mBluetoothAdapter.getScanMode() !=
5.       BluetoothAdapter.SCAN_MODE_CONNECTABLE_DISCOVERABLE) {
6.        //请求可见状态
7.
   Intent discoverableIntent = new Intent(BluetoothAdapter.ACTION_REQUE
   ST_DISCOVERABLE);
8.       //添加附加属性，可见状态的时间
9.
   discoverableIntent.putExtra(BluetoothAdapter.EXTRA_DISCOVERABLE_DU
   RATION, 300);
10.      startActivity(discoverableIntent);
11.      }
12.      }
```

这里首先通过 mBluetoothAdapter.getScanMode()函数取得该蓝牙的扫描模式，然后通过 BluetoothAdapter.ACTION_REQUEST_DISCOVERABLE 设置可见属性，在这里我们加入一个附加属性 BluetoothAdapter.EXTRA_DISCOVERABLE_DURATION，用来设置可见状态的时间，表示在指定的时间中蓝牙处于可见状态，设置好之后通过 startActivity 来执行即可。

这里忧一个需要注意的问题，在链接某个设备之前，我们需要开启一个端口监听，该应用程序将其放在 onResume()函数中来处理了，代码如下：

view plaincopy to clipboardprint?

```
1. @Override
2.       public synchronized void onResume() {
3.       super.onResume();
4.       if(D) Log.e(TAG, "+ ON RESUME +");
5.
6.
   // Performing this check in onResume() covers the case in which BT was
7.       // not enabled during onStart(), so we were paused to enable it...
8.
   // onResume() will be called when ACTION_REQUEST_ENABLE activity ret
   urns.
9.       if (mChatService != null) {
10.      // 如果当前状态为 STATE_NONE，则需要开启蓝牙聊天服务
11.
   if (mChatService.getState() == BluetoothChatService.STATE_NONE) {
```

```
12.     // 开始一个蓝牙聊天服务
13.     mChatService.start();
14.     }
15.     }
16.     }
```

首先检测 mChatService 是否被初始化，然后检测其状态是否为 STATE_NONE，STATE_NONE 表示初始化之后处于等待的状态，当我们在 setupChat 函数中初始时，其实就已经将其状态设置为 STATE_NONE 了（该操作是在 BluetoothChatService 的构造函数中处理的），所以这里就可以通过一个 start 函数来启动一个进程即可，实际上就是启动了一个端口监听进程，当有设备连接时，该监听进程结束，然后转向链接进程，链接之后同样又将转换到一个聊天管理进程，对于这些链接过程和通信模块看来只有下一篇文章介绍了，准备睡觉，明天还得上班！哎...

总结

本文试图完成一个蓝牙聊天程序的分析和实现，但是我发现只要涉及网络通信模块的程序分析过程都比较复杂，可能是因为分析太细的原因，并没有分析完该程序，因此，下一篇文章我们继续分析，同时下一篇要介绍的也是最核心的内容，网络通信过程，希望能得到大家的支持。！

作者介绍

刘子奇（万游数字）

（声明：本网的新闻及文章版权均属 OPhone SDN 网站所有，如需转载请与我们编辑团队联系。任何媒体、网站或个人未经本网书面协议授权，不得进行任何形式的转载。已经取得本网协议授权的媒体、网站，在转载使用时请注明稿件来源。）

接着上一篇没有完成的任务，我们继续分析这个蓝牙聊天程序的实现，本文主要包括以下两个部分的内容：其一，分析扫描设备部分 DeviceListActivity，其二，分析具体的聊天过程的完整通信方案，包括端口监听、链接配对、消息发送和接收等，如果有对上一篇文章不太熟悉的，可以返回去在过一次，这样会有利于本文的理解。

设备扫描（DeviceListActivity）

在上一篇文章的介绍中，当用户点击了扫描按钮之后，则会执行如下代码：

view plaincopy to clipboardprint?

```
1.  // 启动 DeviceListActivity 查看设备并扫描
2.          Intent serverIntent = new Intent(this, DeviceListActivity.class);
3.          startActivityForResult(serverIntent, REQUEST_CONNECT_DEVICE);
```

该代码将跳转到 DeviceListActivity 进行设备的扫描，并且通过 REQUEST_CONNECT_DEVICE 来请求链接扫描到的设备。从 AndroidManifest.xml 文件中我们知道 DeviceListActivity 将为定义为一个对话框的风格，下图是该应用程序中，扫描蓝牙设备的截图。

其中 DeviceListActivity 则为图中对话框部分，其界面的布局如下代码所示。

view plaincopy to clipboardprint?

```
1.  <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
2.          android:orientation="vertical"
3.          android:layout_width="match_parent"
4.          android:layout_height="match_parent"
5.          >
6.          <!-- 已经配对的设备 -->
7.          <TextView android:id="@+id/title_paired_devices"
8.          android:layout_width="match_parent"
9.          android:layout_height="wrap_content"
10.         android:text="@string/title_paired_devices"
11.         android:visibility="gone"
12.         android:background="#666"
13.         android:textColor="#fff"
14.         android:paddingLeft="5dp"
15.         />
16.         <!-- 已经配对的设备信息 -->
17.         <ListView android:id="@+id/paired_devices"
18.         android:layout_width="match_parent"
19.         android:layout_height="wrap_content"
20.         android:stackFromBottom="true"
21.         android:layout_weight="1"
22.         />
23.         <!-- 扫描出来没有经过配对的设备 -->
24.         <TextView android:id="@+id/title_new_devices"
25.         android:layout_width="match_parent"
```

```
26.        android:layout_height="wrap_content"
27.        android:text="@string/title_other_devices"
28.        android:visibility="gone"
29.        android:background="#666"
30.        android:textColor="#fff"
31.        android:paddingLeft="5dp"
32.        />
33.        <!-- 扫描出来没有经过配对的设备信息 -->
34.        <ListView android:id="@+id/new_devices"
35.        android:layout_width="match_parent"
36.        android:layout_height="wrap_content"
37.        android:stackFromBottom="true"
38.        android:layout_weight="2"
39.        />
40.        <!-- 扫描按钮 -->
41.        <Button android:id="@+id/button_scan"
42.        android:layout_width="match_parent"
43.        android:layout_height="wrap_content"
44.        android:text="@string/button_scan"
45.        />
46.        </LinearLayout>
```

该布局整体由一个线性布局 LinearLayout 组成，其中包含了两个 textview 中来显示已经配对的设备和信扫描出来的设备（还没有经过配对）和两个 ListView 分别用于显示已经配对和没有配对的设备的相关信息。按钮则用于执行扫描过程用，整个结构很简单，下面我们开始分析如何编码实现了。

同样开始之前，我们先确定该类中的变量的作用，定义如下：

view plaincopy to clipboardprint?

```
1.   public class DeviceListActivity extends Activity {
2.        // Debugging
3.        private static final String TAG = "DeviceListActivity";
4.        private static final boolean D = true;
5.
6.        // Return Intent extra
7.        public static String EXTRA_DEVICE_ADDRESS = "device_address";
8.
9.        // 蓝牙适配器
10.       private BluetoothAdapter mBtAdapter;
11.       //已经配对的蓝牙设备
12.       private ArrayAdapter<String> mPairedDevicesArrayAdapter;
13.       //新的蓝牙设备
14.       private ArrayAdapter<String> mNewDevicesArrayAdapter;
```

其中 Debugging 部分，同样用于调试，这里定义了一个 EXTRA_DEVICE_ADDRESS，用于在通过 Intent 传递数据时的附加信息，即设备的地址，当扫描出来之后，返回到 BluetoothChat

中的 onActivityResult 函数的 REQUEST_CONNECT_DEVICE 命令，这是我们就需要通过 DeviceListActivity.EXTRA_DEVICE_ADDRESS 来取得该设备的 Mac 地址，因此当我们扫描完成之后在反馈扫描结果时就需要绑定设备地址作为 EXTRA_DEVICE_ADDRESS 的附加值，这和我们上一篇介绍的并不矛盾。另外其他几个变量则分别是本地蓝牙适配器、已经配对的蓝牙列表和扫描出来还没有配对的蓝牙设备列表，稍后我们可以看到对他们的使用。

进入 DeviceListActivity 之后我们首先分析 onCreate，首先通过如下代码对窗口进行了设置：

view plaincopy to clipboardprint?

```
1.  // 设置窗口
2.          requestWindowFeature(Window.FEATURE_INDETERMINATE_PROGRESS);
3.          setContentView(R.layout.device_list);
4.          setResult(Activity.RESULT_CANCELED);
```

这里我们设置了窗口需要带一个进度条，当我们在扫描时就看有很容易的高速用户扫描进度。具体布局则设置为 device_list.xml 也是我们文本第一段代码的内容，接下来首先初始化扫描按钮，代码如下：

view plaincopy to clipboardprint?

```
1.  // 初始化扫描按钮
2.          Button scanButton = (Button) findViewById(R.id.button_scan);
3.          scanButton.setOnClickListener(new OnClickListener() {
4.          public void onClick(View v) {
5.          doDiscovery();
6.          v.setVisibility(View.GONE);
7.          }
8.          });
```

首先取得按钮对象，然后为其设置一个事件监听，当事件触发时就通过 doDiscovery 函数来执行扫描操作即可，具体扫描过程稍后分析。

然后需要初始化用来显示设备的列表和数据源，使用如下代码即可：

view plaincopy to clipboardprint?

```
1.  //初始化 ArrayAdapter，一个是已经配对的设备，一个是新发现的设备
2.
    mPairedDevicesArrayAdapter = new ArrayAdapter<String>(this, R.layout.device_name);

3.
    mNewDevicesArrayAdapter = new ArrayAdapter<String>(this, R.layout.device_name);
4.
5.          // 检测并设置已配对的设备 ListView
6.          ListView pairedListView = (ListView) findViewById(R.id.paired_devices);
7.          pairedListView.setAdapter(mPairedDevicesArrayAdapter);
8.          pairedListView.setOnItemClickListener(mDeviceClickListener);
9.
10.         // 检查并设置行发现的蓝牙设备 ListView
11.         ListView newDevicesListView = (ListView) findViewById(R.id.new_devices);
12.         newDevicesListView.setAdapter(mNewDevicesArrayAdapter);
```

```
13.    newDevicesListView.setOnItemClickListener(mDeviceClickListener);
```

并分别对这些列表中的选项设置了监听 mDeviceClickListener，用来处理，当选择该选项时，就进行链接和配对操作。既然是扫描，我们就需要对扫描的结果进行监控，这里我们构建了一个广播 BroadcastReceiver 来对扫描的结果进行处理，代码如下：

view plaincopy to clipboardprint?

```
1.  // 当一个设备被发现时，需要注册一个广播
2.      IntentFilter filter = new IntentFilter(BluetoothDevice.ACTION_FOUND);
3.      this.registerReceiver(mReceiver, filter);
4.
5.      // 当显示检查完毕的时候，需要注册一个广播
6.      filter = new IntentFilter(BluetoothAdapter.ACTION_DISCOVERY_FINISHED);
7.      this.registerReceiver(mReceiver, filter);
```

这里我们注册到广播 mReceiver 的 IntentFilter 主要包括了发现蓝牙设备（BluetoothDevice.ACTION_FOUND）和扫描结束（BluetoothAdapter.ACTION_DISCOVERY_FINISHED），稍后我们分析如何在 mReceiver 中来处理这些事件。

最后我们需要取得本地蓝牙适配器和一些初始的蓝牙设备数据显示列表进行处理，代码如下：

view plaincopy to clipboardprint?

```
1.  // 得到本地的蓝牙适配器
2.      mBtAdapter = BluetoothAdapter.getDefaultAdapter();
3.
4.      // 得到一个已经匹配到本地适配器的 BluetoothDevice 类的对象集合
5.      Set<BluetoothDevice> pairedDevices = mBtAdapter.getBondedDevices();
6.
7.      // 如果有配对成功的设备则添加到 ArrayAdapter
8.      if (pairedDevices.size() > 0) {
9.      findViewById(R.id.title_paired_devices).setVisibility(View.VISIBLE);
10.     for (BluetoothDevice device : pairedDevices) {
11.
    mPairedDevicesArrayAdapter.add(device.getName() + "\n" + device.getAddress());
12.     }
13.     } else {
14.      //否则添加一个没有被配对的字符串
15.     String noDevices = getResources().getText(R.string.none_paired).toString();
16.     mPairedDevicesArrayAdapter.add(noDevices);
17.     }
```

首先通过蓝牙适配器的 getBondedDevices 函数取得已经配对的蓝牙设备，并将其添加到 mPairedDevicesArrayAdapter 数据源中，会显示到 pairedListView 列表视图中，如果没有已经配对的蓝牙设备，则显示一个 R.string.none_paired 字符串表示目前没有配对成功的设备。

onDestroy 函数中会制定销毁操作，主要包括蓝牙适配器和广播的注销操作，代码如下：

view plaincopy to clipboardprint?

```
1.  @Override
2.      protected void onDestroy() {
3.      super.onDestroy();
4.      // 确保我们没有发现，检测设备
5.      if (mBtAdapter != null) {
6.      mBtAdapter.cancelDiscovery();
7.      }
8.      // 卸载所注册的广播
9.      this.unregisterReceiver(mReceiver);
10.     }
```

对于蓝牙适配器的取消方式则调用 cancelDiscovery()函数即可，卸载 mReceiver 则需要调用 unregisterReceiver 即可。

做好初始化工作之后，下面我们开始分析扫描函数 doDiscovery()，其扫描过程的实现很就简单，代码如下：

view plaincopy to clipboardprint?

```
1.  /**
2.      * 请求能被发现的设备
3.      */
4.      private void doDiscovery() {
5.      if (D) Log.d(TAG, "doDiscovery()");
6.
7.      // 设置显示进度条
8.      setProgressBarIndeterminateVisibility(true);
9.      // 设置 title 为扫描状态
10.     setTitle(R.string.scanning);
11.
12.     // 显示新设备的子标题
13.     findViewById(R.id.title_new_devices).setVisibility(View.VISIBLE);
14.
15.     // 如果已经在请求现实了，那么就先停止
16.     if (mBtAdapter.isDiscovering()) {
17.     mBtAdapter.cancelDiscovery();
18.     }
19.
20.     // 请求从蓝牙适配器得到能够被发现的设备
21.     mBtAdapter.startDiscovery();
22.     }
```

首先通过 setProgressBarIndeterminateVisibility 将进度条设置为显示状态，设置标题 title 为 R.string.scanning 字符串，表示正在扫描中，代码中所说的新设备的子标题，其实就是上面我们所说的扫描到的没有经过配对的设备的 title，对应于 R.id.title_new_devices。扫描之前我们首先通过 isDiscovering 函数检测当前是否正在扫描，如果正在扫描则调用 cancelDiscovery 函数来取消当前的扫描，最后调用 startDiscovery 函数开始执行扫描操作。

现在已经开始扫描了，下面我们就需要对扫描过程进行监控和对扫描的结果进行处理。

即我们所定义的广播 mReceiver，其实现如下所示。

view plaincopy to clipboardprint?

```
1. //监听扫描蓝牙设备
2.     private final BroadcastReceiver mReceiver = new BroadcastReceiver() {
3.     @Override
4.     public void onReceive(Context context, Intent intent) {
5.     String action = intent.getAction();
6.     // 当发现一个设备时
7.     if (BluetoothDevice.ACTION_FOUND.equals(action)) {
8.     // 从 Intent 得到蓝牙设备对象
9.
   BluetoothDevice device = intent.getParcelableExtra(BluetoothDevice.EXTRA_DEVICE);
10.    // 如果已经配对，则跳过，因为他已经在设备列表中了
11.    if (device.getBondState() != BluetoothDevice.BOND_BONDED) {
12.     //否则添加到设备列表
13.    mNewDevicesArrayAdapter.add(device.getName() + "\n" + device.getAddress());
14.    }
15.    // 当扫描完成之后改变 Activity 的 title
16.    } else if (BluetoothAdapter.ACTION_DISCOVERY_FINISHED.equals(action)) {
17.     //设置进度条不显示
18.    setProgressBarIndeterminateVisibility(false);
19.    //设置 title
20.    setTitle(R.string.select_device);
21.    //如果计数为 0，则表示没有发现蓝牙
22.    if (mNewDevicesArrayAdapter.getCount() == 0) {
23.    String noDevices = getResources().getText(R.string.none_found).toString();
24.    mNewDevicesArrayAdapter.add(noDevices);
25.    }
26.    }
27.    }
28.    };
```

其中我们通过 intent.getAction()可以取得一个动作，然后判断如果动作为 BluetoothDevice.ACTION_FOUND，则表示发现一个蓝牙设备，然后通过 BluetoothDevice.EXTRA_DEVICE 常量可以取得 Intent 中的蓝牙设备对象（BluetoothDevice），然后通过条件"device.getBondState() != BluetoothDevice.BOND_BONDED"来判断设备是否配对，如果没有配对则添加到行设备列表数据源 mNewDevicesArrayAdapter 中，另外，当我们取得的动作为 BluetoothAdapter.ACTION_DISCOVERY_FINISHED，则表示扫描过程完毕，这时首先需要设置进度条不现实，并且设置窗口的标题为选择一个设备（R.string.select_device）。当然如果扫描完成之后没有发现新的设备，则添加一个没有发现新的设备字符串（R.string.none_found）到 mNewDevicesArrayAdapter 中。

最后，扫描界面上还有一个按钮，其监听 mDeviceClickListener 的实现如下：

view plaincopy to clipboardprint?

```
1. // ListViews 中所有设备的点击事件监听
```

```
2.          private OnItemClickListener mDeviceClickListener = new OnItemClickListener() {
3.          public void onItemClick(AdapterView<?> av, View v, int arg2, long arg3) {
4.          // 取消检测扫描发现设备的过程,因为内非常耗费资源
5.          mBtAdapter.cancelDiscovery();
6.
7.          // 得到 mac 地址
8.          String info = ((TextView) v).getText().toString();
9.          String address = info.substring(info.length() - 17);
10.
11.         // 创建一个包括 Mac 地址的 Intent 请求
12.         Intent intent = new Intent();
13.         intent.putExtra(EXTRA_DEVICE_ADDRESS, address);
14.
15.         // 设置 result 并结束 Activity
16.         setResult(Activity.RESULT_OK, intent);
17.         finish();
18.         }
19.         };
```

当用户点击该按钮时，首先取消扫描进程，因为扫描过程是一个非常耗费资源的过程，然后去的设备的 mac 地址，构建一个 Intent 对象，通过附加数据 EXTRA_DEVICE_ADDRESS 将 mac 地址传递到 BluetoothChat 中，然后调用 finish 来结束该界面。这时就会回到上一篇文章我们介绍的 BluetoothChat 中的 onActivityResult 函数中去执行请求代码为 REQUEST_CONNECT_DEVICE 的片段，用来连接一个设备。

BluetoothChatService

对于设备的监听，连接管理都将由 REQUEST_CONNECT_DEVICE 来实现，其中又包括三个主要部分，三个进程分贝是：请求连接的监听线程（AcceptThread）、连接一个设备的进程（ConnectThread）、连接之后的管理进程（ConnectedThread）。同样我们先熟悉一下该类的成员变量的作用，定义如下：

view plaincopy to clipboardprint?

```
1.   // Debugging
2.          private static final String TAG = "BluetoothChatService";
3.          private static final boolean D = true;
4.
5.          //当创建 socket 服务时的 SDP 名称
6.          private static final String NAME = "BluetoothChat";
7.
8.          // 应用程序的唯一 UUID
9.
    private static final UUID MY_UUID = UUID.fromString("fa87c0d0-afac-11de-8a39-0800200c9a66");
10.
11.         // 本地蓝牙适配器
12.         private final BluetoothAdapter mAdapter;
```

```
13.     //Handler
14.     private final Handler mHandler;
15.     //请求链接的监听线程
16.     private AcceptThread mAcceptThread;
17.     //链接一个设备的线程
18.     private ConnectThread mConnectThread;
19.     //已经链接之后的管理线程
20.     private ConnectedThread mConnectedThread;
21.     //当前的状态
22.     private int mState;
23.
24.     // 各种状态
25.     public static final int STATE_NONE = 0;
26.     public static final int STATE_LISTEN = 1;
27.     public static final int STATE_CONNECTING = 2;
28.     public static final int STATE_CONNECTED = 3;
```

Debugging 为调试相关，NAME 是当我们在创建一个 socket 监听服务时的一个 SDP 名称，另外还包括一个状态变量 mState，其值则分别是下面的"各种状态"部分，另外还有一个本地蓝牙适配器和三个不同的进程对象，由此可见，本地蓝牙适配器的确是任何蓝牙操作的基础对象，下面我们会分别介绍这些进程的实现。

首先是初始化操作，即构造函数，代码如下：

view plaincopy to clipboardprint?

```
1.  public BluetoothChatService(Context context, Handler handler) {
2.       //得到本地蓝牙适配器
3.      mAdapter = BluetoothAdapter.getDefaultAdapter();
4.      //设置状态
5.      mState = STATE_NONE;
6.      //设置 Handler
7.      mHandler = handler;
8.      }
```

取得本地蓝牙适配器、设置状态为 STATE_NONE，设置传递进来的 mHandler。接下来需要控制当状态改变之后，我们需要通知 UI 界面也同时更改状态，下面是得到状态和设置状态的实现部分，如下：

view plaincopy to clipboardprint?

```
1.  private synchronized void setState(int state) {
2.      if (D) Log.d(TAG, "setState() " + mState + " -> " + state);
3.      mState = state;
4.
5.      // 状态更新之后 UI Activity 也需要更新
6.
    mHandler.obtainMessage(BluetoothChat.MESSAGE_STATE_CHANGE, state, -1).sendToTarget();
7.      }
8.
```

```
9.      public synchronized int getState() {
10.     return mState;
11.     }
```

得到状态没有什么特别的，关键在于设置状态之后需要通过 obtainMessage 来发送一个消息到 Handler，通知 UI 界面也同时更新其状态，对应的 Handler 的实现则位于 BluetoothChat 中的 private final Handler mHandler = new Handler()部分，从上面的代码中，我们可以看到关于状态更改的之后会发送一个 BluetoothChat.MESSAGE_STATE_CHANGE 消息到 UI 线程中，下面我们看一下 UI 线程中如何处理这些消息的，代码如下：

view plaincopy to clipboardprint?

```
1.  case MESSAGE_STATE_CHANGE:
2.      if(D) Log.i(TAG, "MESSAGE_STATE_CHANGE: " + msg.arg1);
3.      switch (msg.arg1) {
4.      case BluetoothChatService.STATE_CONNECTED:
5.       //设置状态为已经链接
6.      mTitle.setText(R.string.title_connected_to);
7.      //添加设备名称
8.      mTitle.append(mConnectedDeviceName);
9.      //清理聊天记录
10.     mConversationArrayAdapter.clear();
11.     break;
12.     case BluetoothChatService.STATE_CONNECTING:
13.      //设置正在链接
14.     mTitle.setText(R.string.title_connecting);
15.     break;
16.     case BluetoothChatService.STATE_LISTEN:
17.     case BluetoothChatService.STATE_NONE:
18.      //处于监听状态或者没有准备状态，则显示没有链接
19.     mTitle.setText(R.string.title_not_connected);
20.     break;
21.     }
22.     break;
```

可以看出，当不同的状态在更改之后会进行不同的设置，但是大多数都是根据不同的状态设置显示了不同的 title，当已经链接（STATE_CONNECTED）之后,设置了标题为链接的设备名，并同时还 mConversationArrayAdapter 进行了清除操作，即清除聊天记录。

现在，初始化操作已经完成了，下面我们可以调用 start 函数来开启一个服务进程了，也即是在 BluetoothChat 中的 onResume 函数中所调用的 start 操作，其具体实现如下：

view plaincopy to clipboardprint?

```
1.  public synchronized void start() {
2.      if (D) Log.d(TAG, "start");
3.
4.      // 取消任何线程视图建立一个连接
5.      if (mConnectThread != null) {mConnectThread.cancel(); mConnectThread = null;}
6.
7.      // 取消任何正在运行的链接
```

```
8.
    if (mConnectedThread != null) {mConnectedThread.cancel(); mConnectedThread = null;}

9.
10.         // 启动 AcceptThread 线程来监听 BluetoothServerSocket
11.         if (mAcceptThread == null) {
12.         mAcceptThread = new AcceptThread();
13.         mAcceptThread.start();
14.         }
15.         //设置状态为监听，，等待链接
16.         setState(STATE_LISTEN);
17.         }
```

操作过程很简单，首先取消另外两个进程，新建一个 AcceptThread 进程，并启动 AcceptThread 进程，最后设置状态变为监听(STATE_LISTEN)，这时 UI 界面的 title 也将更新为监听状态，即等待设备的连接。关于 AcceptThread 的具体实现如下所示。

view plaincopy to clipboardprint?

```
1.   private class AcceptThread extends Thread {
2.          // 本地 socket 服务
3.          private final BluetoothServerSocket mmServerSocket;
4.
5.          public AcceptThread() {
6.          BluetoothServerSocket tmp = null;
7.
8.          // 创建一个新的 socket 服务监听
9.          try {
10.         tmp = mAdapter.listenUsingRfcommWithServiceRecord(NAME, MY_UUID);
11.         } catch (IOException e) {
12.         Log.e(TAG, "listen() failed", e);
13.         }
14.         mmServerSocket = tmp;
15.         }
16.
17.         public void run() {
18.         if (D) Log.d(TAG, "BEGIN mAcceptThread" + this);
19.         setName("AcceptThread");
20.         BluetoothSocket socket = null;
21.
22.         // 如果当前没有链接则一直监听 socket 服务
23.         while (mState != STATE_CONNECTED) {
24.         try {
25.          //如果有请求链接，则接受
26.          //这是一个阻塞调用，将之返回链接成功和一个异常
27.         socket = mmServerSocket.accept();
28.         } catch (IOException e) {
```

```
29.	Log.e(TAG, "accept() failed", e);
30.	break;
31.	}
32.
33.	// 如果接受了一个链接
34.	if (socket != null) {
35.	synchronized (BluetoothChatService.this) {
36.	switch (mState) {
37.	case STATE_LISTEN:
38.	case STATE_CONNECTING:
39.	// 如果状态为监听或者正在链接中,, 则调用 connected 来链接
40.	connected(socket, socket.getRemoteDevice());
41.	break;
42.	case STATE_NONE:
43.	case STATE_CONNECTED:
44.	// 如果为没有准备或者已经链接，这终止该 socket
45.	try {
46.	socket.close();
47.	} catch (IOException e) {
48.	Log.e(TAG, "Could not close unwanted socket", e);
49.	}
50.	break;
51.	}
52.	}
53.	}
54.	}
55.	if (D) Log.i(TAG, "END mAcceptThread");
56.	}
57.	//关闭 BluetoothServerSocket
58.	public void cancel() {
59.	if (D) Log.d(TAG, "cancel " + this);
60.	try {
61.	mmServerSocket.close();
62.	} catch (IOException e) {
63.	Log.e(TAG, "close() of server failed", e);
64.	}
65.	}
66.	}
```

首先通过 listenUsingRfcommWithServiceRecord 创建一个 socket 服务，用来监听设备的连接，当进程启动之后直到有设备连接时，这段时间都将通过 accept 来监听和接收一个连接请求，如果连接无效则调用 close 来关闭即可，如果连接有效则调用 connected 进入连接进程，进入连接进程之后会取消当前的监听进程，取消过程则直接调用 cancel 通过 mmServerSocket.close()来关闭即可。下面我们分析连接函数 connect 的实现，如下：

view plaincopy to clipboardprint?

```
1.  public synchronized void connect(BluetoothDevice device) {
2.      if (D) Log.d(TAG, "connect to: " + device);
3.  
4.      // 取消任何链接线程，视图建立一个链接
5.      if (mState == STATE_CONNECTING) {
6.      if (mConnectThread != null) {mConnectThread.cancel(); mConnectThread = null;}
7.      }
8.  
9.      // 取消任何正在运行的线程
10. 
    if (mConnectedThread != null) {mConnectedThread.cancel(); mConnectedThread = null;}

11. 
12.     // 启动一个链接线程链接指定的设备
13.     mConnectThread = new ConnectThread(device);
14.     mConnectThread.start();
15.     setState(STATE_CONNECTING);
16.     }
```

同样，首先关闭其他两个进程，然后新建一个 ConnectThread 进程，并启动，通知 UI 界面状态更改为正在连接的状态（STATE_CONNECTING）。具体的连接进程由 ConnectThread 来实现，如下：

view plaincopy to clipboardprint?

```
1.  private class ConnectThread extends Thread {
2.       //蓝牙 Socket
3.      private final BluetoothSocket mmSocket;
4.      //蓝牙设备
5.      private final BluetoothDevice mmDevice;
6.  
7.      public ConnectThread(BluetoothDevice device) {
8.      mmDevice = device;
9.      BluetoothSocket tmp = null;
10. 
11.     //得到一个给定的蓝牙设备的 BluetoothSocket
12.     try {
13.     tmp = device.createRfcommSocketToServiceRecord(MY_UUID);
14.     } catch (IOException e) {
15.     Log.e(TAG, "create() failed", e);
16.     }
17.     mmSocket = tmp;
18.     }
19. 
20.     public void run() {
```

```
21.	Log.i(TAG, "BEGIN mConnectThread");
22.	setName("ConnectThread");
23.	
24.	// 取消可见状态，将会进行链接
25.	mAdapter.cancelDiscovery();
26.	
27.	// 创建一个 BluetoothSocket 链接
28.	try {
29.	//同样是一个阻塞调用，返回成功和异常
30.	mmSocket.connect();
31.	} catch (IOException e) {
32.	 //链接失败
33.	connectionFailed();
34.	// 如果异常则关闭 socket
35.	try {
36.	mmSocket.close();
37.	} catch (IOException e2) {
38.	Log.e(TAG, "unable to close() socket during connection failure", e2);
39.	}
40.	// 重新启动监听服务状态
41.	BluetoothChatService.this.start();
42.	return;
43.	}
44.	
45.	// 完成则重置 ConnectThread
46.	synchronized (BluetoothChatService.this) {
47.	mConnectThread = null;
48.	}
49.	
50.	// 开启 ConnectedThread（正在运行中...）线程
51.	connected(mmSocket, mmDevice);
52.	}
53.	//取消链接线程 ConnectThread
54.	public void cancel() {
55.	try {
56.	mmSocket.close();
57.	} catch (IOException e) {
58.	Log.e(TAG, "close() of connect socket failed", e);
59.	}
60.	}
61.	}
```

在创建该进程时，就已经知道当前需要被连接的蓝牙设备，然后通过 createRfcommSocketToServiceRecord 可以构建一个蓝牙设备的 BluetoothSocket 对象，当进入连接状态时，就可以调用 cancelDiscovery 来取消蓝牙的可见状态，然后通过调用 connect 函

数进行链接操作，如果出现异常则表示链接失败，则调用 connectionFailed 函数通知 UI 进程更新界面的显示为链接失败状态，然后关闭 BluetoothSocket，调用 start 函数重新开启一个监听服务 AcceptThread，对于链接失败的处理实现如下：

view plaincopy to clipboardprint?

```
1.  private void connectionFailed() {
2.          setState(STATE_LISTEN);
3.
4.          // 发送链接失败的消息到 UI 界面
5.          Message msg = mHandler.obtainMessage(BluetoothChat.MESSAGE_TOAST);
6.          Bundle bundle = new Bundle();
7.          bundle.putString(BluetoothChat.TOAST, "Unable to connect device");
8.          msg.setData(bundle);
9.          mHandler.sendMessage(msg);
10.     }
```

首先更改状态为 STATE_LISTEN，然后发送一个 Message 带 UI 界面，通知 UI 更新，显示一个 Toast 告知用户，当 BluetoothChat 中的 mHandler 接收到 BluetoothChat.TOAST 消息时，就会直接更新 UI 界面的显示，如果连接成功则将调用 connected 函数进入连接管理进程，其实现如下：

view plaincopy to clipboardprint?

```
1.  public synchronized void connected(BluetoothSocket socket, BluetoothDevice device) {

2.          if (D) Log.d(TAG, "connected");
3.
4.          // 取消 ConnectThread 链接线程
5.          if (mConnectThread != null) {mConnectThread.cancel(); mConnectThread = null;}
6.
7.          // 取消所有正在链接的线程
8.
    if (mConnectedThread != null) {mConnectedThread.cancel(); mConnectedThread = null;}

9.
10.         // 取消所有的监听线程，因为我们已经链接了一个设备
11.         if (mAcceptThread != null) {mAcceptThread.cancel(); mAcceptThread = null;}
12.
13.         // 启动 ConnectedThread 线程来管理链接和执行翻译
14.         mConnectedThread = new ConnectedThread(socket);
15.         mConnectedThread.start();
16.
17.         // 发送链接的设备名称到 UI Activity 界面
18.
    Message msg = mHandler.obtainMessage(BluetoothChat.MESSAGE_DEVICE_NAME);
19.         Bundle bundle = new Bundle();
20.         bundle.putString(BluetoothChat.DEVICE_NAME, device.getName());
21.         msg.setData(bundle);
```

22. mHandler.sendMessage(msg);
23. //状态变为已经链接，即正在运行中
24. setState(STATE_CONNECTED);
25. }

首先，关闭所有的进程，构建一个 ConnectedThread 进程，并准备一个 Message 消息，就设备名称（BluetoothChat.DEVICE_NAME）也发送到 UI 进程，因为 UI 进程需要显示当前连接的设备名称，当 UI 进程收到 BluetoothChat.MESSAGE_DEVICE_NAME 消息时就会更新相应的 UI 界面，就是设置窗口的 title，这里我们就不贴出代码了，下面我们分析一下 ConnectedThread 的实现，代码如下：

view plaincopy to clipboardprint?

```
1. private class ConnectedThread extends Thread {
2.     //BluetoothSocket
3.     private final BluetoothSocket mmSocket;
4.     //输入输出流
5.     private final InputStream mmInStream;
6.     private final OutputStream mmOutStream;
7.
8.     public ConnectedThread(BluetoothSocket socket) {
9.     Log.d(TAG, "create ConnectedThread");
10.    mmSocket = socket;
11.    InputStream tmpIn = null;
12.    OutputStream tmpOut = null;
13.
14.    // 得到 BluetoothSocket 的输入输出流
15.    try {
16.    tmpIn = socket.getInputStream();
17.    tmpOut = socket.getOutputStream();
18.    } catch (IOException e) {
19.    Log.e(TAG, "temp sockets not created", e);
20.    }
21.
22.    mmInStream = tmpIn;
23.    mmOutStream = tmpOut;
24.    }
25.
26.    public void run() {
27.    Log.i(TAG, "BEGIN mConnectedThread");
28.    byte[] buffer = new byte[1024];
29.    int bytes;
30.
31.    // 监听输入流
32.    while (true) {
33.    try {
34.    // 从输入流中读取数据
```

```
35.     bytes = mmInStream.read(buffer);
36.
37.     // 发送一个消息到 UI 线程进行更新
38.     mHandler.obtainMessage(BluetoothChat.MESSAGE_READ, bytes, -1, buffer)
39.     .sendToTarget();
40.     } catch (IOException e) {
41.      //出现异常，则链接丢失
42.     Log.e(TAG, "disconnected", e);
43.     connectionLost();
44.     break;
45.     }
46.     }
47.     }
1.
        /**
2.      * 写入药发送的消息
3.      * @param buffer  The bytes to write
4.      */
5.      public void write(byte[] buffer) {
6.      try {
7.      mmOutStream.write(buffer);
8.
9.      // 将写的消息同时传递给 UI 界面
10.     mHandler.obtainMessage(BluetoothChat.MESSAGE_WRITE, -1, -1, buffer)
11.     .sendToTarget();
12.     } catch (IOException e) {
13.     Log.e(TAG, "Exception during write", e);
14.     }
15.     }
16.     //取消 ConnectedThread 链接管理线程
17.     public void cancel() {
18.     try {
19.     mmSocket.close();
20.     } catch (IOException e) {
21.     Log.e(TAG, "close() of connect socket failed", e);
22.     }
23.     }
24.     }
```

连接之后的主要操作就是发送和接收聊天消息了，因为需要通过其输入（出）流来操作具体信息，进程会一直从输入流中读取信息，并通过 obtainMessage 函数将读取的信息以 BluetoothChat.MESSAGE_READ 命令发送到 UI 进程，到 UI 进程收到是，就需要将其显示到消息列表之中，同时对于发送消息，需要实行写操作 write，其操作就是将要发送的消息写入到输出流 mmOutStream 中，并且以 BluetoothChat.MESSAGE_WRITE 命令的方式发送到 UI 进程中，进行同步更新，如果在读取消息时失败或者产生了异常，则表示连接丢失，这是就调

用 connectionLost 函数来处理连接丢失，代码如下：

view plaincopy to clipboardprint?

```
1.  private void connectionLost() {
2.          setState(STATE_LISTEN);
3.  
4.          // 发送失败消息到 UI 界面
5.          Message msg = mHandler.obtainMessage(BluetoothChat.MESSAGE_TOAST);
6.          Bundle bundle = new Bundle();
7.          bundle.putString(BluetoothChat.TOAST, "Device connection was lost");
8.          msg.setData(bundle);
9.          mHandler.sendMessage(msg);
10.     }
```

操作同样简单，首先改变状态为 STATE_LISTEN，然后 BluetoothChat.MESSAGE_TOAST 命令发送一个消息 Message 到 UI 进程，通知 UI 进程更新显示画面即可。对于写操作，是调用了 BluetoothChatService.write 来实现，其实现代码如下：

view plaincopy to clipboardprint?

```
1.  //写入自己要发送出来的消息
2.          public void write(byte[] out) {
3.          // Create temporary object
4.          ConnectedThread r;
5.          // Synchronize a copy of the ConnectedThread
6.          synchronized (this) {
7.           //判断是否处于已经链接状态
8.          if (mState != STATE_CONNECTED) return;
9.          r = mConnectedThread;
10.         }
11.         // 执行写
12.         r.write(out);
13.         }
```

其实就是检测，当前的状态是否处于已经链接状态 STATE_CONNECTED，然后调用 ConnectedThread 进程中的 write 操作，来完成消息的发送。因此这时我们可以回过头来看 BluetoothChat 中的 sendMessage 的实现了，如下所示：

view plaincopy to clipboardprint?

```
1.  private void sendMessage(String message) {
2.          // 检查是否处于连接状态
3.          if (mChatService.getState() != BluetoothChatService.STATE_CONNECTED) {
4.          Toast.makeText(this, R.string.not_connected, Toast.LENGTH_SHORT).show();
5.          return;
6.          }
7.  
8.          // 如果输入的消息不为空才发送，否则不发送
9.          if (message.length() > 0) {
10.         // Get the message bytes and tell the BluetoothChatService to write
11.         byte[] send = message.getBytes();
```

```
12.        mChatService.write(send);
13.
14.        // Reset out string buffer to zero and clear the edit text field
15.        mOutStringBuffer.setLength(0);
16.        mOutEditText.setText(mOutStringBuffer);
17.        }
18.        }
```

同样首先检测了当前的状态是否为已经连接状态，然后对要发送的消息是否为 null 进行了判断，如果为空则不需要发送，否则调用 mChatService.write（即上面所说的 ConnectedThread 中的 wirte 操作）来发送消息。然后一个小的细节就是设置编辑框的内容为 null 即可。最后我们可以看一下在 BluetoothChat 中如何处理这些接收到的消息，主要位于 mHandler 中的 handleMessage 函数中，对于状态改变的消息我们已经分析过了，下面是其他几个消息的处理：

view plaincopy to clipboardprint?

```
1.    case MESSAGE_WRITE:
2.        byte[] writeBuf = (byte[]) msg.obj;
3.        // 将自己写入的消息也显示到会话列表中
4.        String writeMessage = new String(writeBuf);
5.        mConversationArrayAdapter.add("Me:  " + writeMessage);
6.        break;
7.        case MESSAGE_READ:
8.        byte[] readBuf = (byte[]) msg.obj;
9.        // 取得内容并添加到聊天对话列表中
10.       String readMessage = new String(readBuf, 0, msg.arg1);
11.       mConversationArrayAdapter.add(mConnectedDeviceName+":  " + readMessage);
12.       break;
13.       case MESSAGE_DEVICE_NAME:
14.       // 保存链接的设备名称，并显示一个 toast 提示
15.       mConnectedDeviceName = msg.getData().getString(DEVICE_NAME);
16.       Toast.makeText(getApplicationContext(), "Connected to "
17.       + mConnectedDeviceName, Toast.LENGTH_SHORT).show();
18.       break;
19.       case MESSAGE_TOAST:
20.        //处理链接(发送)失败的消息
21.       Toast.makeText(getApplicationContext(), msg.getData().getString(TOAST),
22.       Toast.LENGTH_SHORT).show();
23.       break;
```

分别是读取消息和写消息（发送消息），对于一些信息提示消息 MESSAGE_TOAST，则通过 Toast 显示出来即可。如果消息是设备名称 MESSAGE_DEVICE_NAME，则提示用户当前连接的设备的名称。对于写消息（MESSAGE_WRITE）和读消息（MESSAGE_READ）我们就不重复了，大家看看代码都已经加入了详细的注释了。

最后当我们在需要停止这些进程时就看有直接调用 stop 即可，具体实现如下：

view plaincopy to clipboardprint?

```
1.    //停止所有的线程
```

```
2.        public synchronized void stop() {
3.        if (D) Log.d(TAG, "stop");
4.        if (mConnectThread != null) {mConnectThread.cancel(); mConnectThread = null;}
5.
    if (mConnectedThread != null) {mConnectedThread.cancel(); mConnectedThread = null;}

6.        if (mAcceptThread != null) {mAcceptThread.cancel(); mAcceptThread = null;}
7.        //状态设置为准备状态
8.        setState(STATE_NONE);
9.        }
```

分别检测三个进程是否为 null，然后调用各自的 cancel 函数来取消进程，最后不要忘记将状态恢复到 STATE_NONE 即可。

总结

终于完成了对蓝牙聊天程序的实现和分析，该示例程序比较全面，基本上包括了蓝牙编程的各个方面，希望通过这几篇文章的问题，能够帮助大家理解在 Ophone 平台上进行蓝牙编程，同时将蓝牙技术运用到其他应用程序中实现应用程序的网络化，联机性。或许你有更多的用处。