BlueTooth连接速悦得obd设备的总结

2014年03月17日 14:06:57 yaya_soft 阅读数 6730 更多

前段时间项目中需要使用手机蓝牙去连接第三方的设备，读取设备中的信息，开始搞的晕头雾水的。。还好最终搞定了。写篇帖子来记录一下这段苦逼蓝牙旅程

android官网提供了一个基于蓝牙的聊天案例。、有人已经把demo抽出来做了详尽的分析http://trylovecatch.iteye.com/blog/1937036。。。我这篇文章主要来分析如何同第三三方设备通过蓝牙来进行信息交互

开发步骤

1: 添加蓝牙操作权限

```
<uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />
 <uses-permission android:name="android.permission.BLUETOOTH" />
```

2：获取BluttoothAdapter 并检测设备是否支持蓝牙

```
/**
         * 确认设备是否支持蓝牙
         * 如果getDefaultAdapter()返回null,则这个设备不支持蓝牙
         */
        mBluetoothAdapter = BluetoothAdapter.getDefaultAdapter();

        // If the adapter is null, then Bluetooth is not supported
        //手机不支持蓝牙
        if (mBluetoothAdapter == null) {
            Toast.makeText(this, "Bluetooth is not available", Toast.LENGTH_LONG).show();
            finish();
            return;
        }
```

在api中解释了何为BlueToothAdapter--> Represents the local device Bluetooth adapter其实就代表本地的蓝牙

3：判断蓝牙是否可用

```
/**
         * 确定蓝牙能够使用。
         * 通过isEnabled()来检查蓝牙当前是否可用。
         * 如果这个方法返回false,则蓝牙不能够使用。这个时候 就请求蓝牙使用，通过intent来请求
                * 在onActivityResult () 里面判断
         */
        if (!mBluetoothAdapter.isEnabled()) {
            Intent enableIntent = new Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
            startActivityForResult(enableIntent, REQUEST_ENABLE_BT);
        // Otherwise, setup the chat session
        }
```

4：请求周围的蓝牙

```
Intent serverIntent = new Intent(this, DeviceListActivity.class);
startActivityForResult(serverIntent, REQUEST_CONNECT_DEVICE);
```

返回的结果在onActivityResult中获取【主要是找到设备蓝牙的address】

```
        @Override
        protected void onActivityResult(int requestCode, int resultCode, Intent data) {
                // TODO Auto-generated method stub
                switch (requestCode) {
                case REQUEST_CONNECT_DEVICE:
                                if (resultCode == Activity.RESULT_OK) {
                        // Get the device MAC address
                        String address = data.getExtras()
                                                .getString(DeviceListActivity.EXTRA_DEVICE_ADDRESS);
                        mBluetoothSet.ConnectDevices(address);
                    }
                        break;

                default:
                        break;
                }
                super.onActivityResult(requestCode, resultCode, data);
        }
```

5:在请求远程蓝牙之前，【onpause】方法中启动远程设备蓝牙服务【作为服务端】

```
/**
    * This thread runs while listening for incoming connections. It behaves
    * like a server-side client. It runs until a connection is accepted
    * (or until cancelled).
    */
   private class AcceptThread extends Thread {
        // The local server socket
        private final BluetoothServerSocket mmServerSocket;

        public AcceptThread() {
            BluetoothServerSocket tmp = null;

            // Create a new listening server socket
            try {
                tmp = mAdapter.listenUsingRfcommWithServiceRecord(NAME, MY_UUID);
            } catch (IOException e) {
                Log.e(TAG, "listen() failed", e);
            }
            mmServerSocket = tmp;
        }

        public void run() {
            if (D) Log.d(TAG, "BEGIN mAcceptThread" + this);
            setName("AcceptThread");
            BluetoothSocket socket = null;

            // Listen to the server socket if we're not connected
            while (mState != STATE_CONNECTED) {
                try {
                    // This is a blocking call and will only return on a
                    // successful connection or an exception
                    socket = mmServerSocket.accept();
                } catch (IOException e) {
                    Log.e(TAG, "accept() failed", e);
                    break;
                }

                // If a connection was accepted
                if (socket != null) {
```

```
                synchronized (BluetoothService.this) {                    switch (mState) {
                    case STATE_LISTEN:
                    case STATE_CONNECTING:
                        // Situation normal. Start the connected thread.
                        connected(socket, socket.getRemoteDevice());
                        break;
                    case STATE_NONE:
                    case STATE_CONNECTED:
                        // Either not ready or already connected. Terminate new socket.
                        try {
                            socket.close();
                        } catch (IOException e) {
                            Log.e(TAG, "Could not close unwanted socket", e);
                        }
                        break;
                    }
                }
            }
        }
        if (D) Log.i(TAG, "END mAcceptThread");
    }

    public void cancel() {
        if (D) Log.d(TAG, "cancel " + this);
        try {
            mmServerSocket.close();
        } catch (IOException e) {
            Log.e(TAG, "close() of server failed", e);
        }
    }
}
```

6：请求服务端蓝牙设备列表

```
package com.thread.est527.bluetooth;

import java.util.ArrayList;
import java.util.List;

import com.thread.est527.R;

import android.app.Activity;
import android.app.ProgressDialog;
import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.DialogInterface;
import android.content.Intent;
import android.content.IntentFilter;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.view.Window;
import android.view.View.OnClickListener;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.Button;
import android.widget.ListView;
import android.widget.TextView;
import android.widget.AdapterView.OnItemClickListener;
```

```
    /**
 * This Activity appears as a dialog. It lists any paired devices and devices
 * detected in the area after discovery. When a device is chosen by the user,
 * the MAC address of the device is sent back to the parent Activity in the
 * result Intent.
 */
public class DeviceListActivity extends Activity
{
        // Debugging
        private static final String TAG = "DeviceListActivity";
        private static final boolean D = true;

        // Return Intent extra
        public static String EXTRA_DEVICE_ADDRESS = "device_address";

        // Member fields
        private BluetoothAdapter mBtAdapter;
        private ArrayAdapter<String> mNewDevicesArrayAdapter;
        List<String> lstDevices = new ArrayList<String>();
        private static Boolean hasDevices;
        private ProgressDialog mpDialog = null;


        @Override
        protected void onCreate(Bundle savedInstanceState)
        {
                super.onCreate(savedInstanceState);

                // Setup the window
                requestWindowFeature(Window.FEATURE_INDETERMINATE_PROGRESS);
                setContentView(R.layout.activity_device_list);

                // Set result CANCELED incase the user backs out
                setResult(Activity.RESULT_CANCELED);

                // Initialize the button to perform device discovery
                Button scanButton = (Button) findViewById(R.id.button_scan);
                scanButton.setOnClickListener(new OnClickListener()
                {
                        public void onClick(View v)
                        {
                                lstDevices.clear();
                                doDiscovery();
                                //v.setVisibility(View.GONE);
                        }
                });

                // Initialize array adapters. One for already paired devices and
                // one for newly discovered devices
                mNewDevicesArrayAdapter = new ArrayAdapter<String>(this,
                                R.layout.device_name,lstDevices);

                // Find and set up the ListView for newly discovered devices
                ListView newDevicesListView = (ListView) findViewById(R.id.new_devices);
                newDevicesListView.setAdapter(mNewDevicesArrayAdapter);
                newDevicesListView.setOnItemClickListener(mDeviceClickListener);

                // Register for broadcasts when a device is discovered
                IntentFilter found_filter = new IntentFilter(BluetoothDevice.ACTION_FOUND);
                this.registerReceiver(mReceiver, found_filter);

                // Register for broadcasts when discovery has finished
                IntentFilter discovery_filter = new IntentFilter(BluetoothAdapter.ACTION_DISCOVERY_FINISHED);
```

```
        this.registerReceiver(mReceiver, discovery_filter);

        // Get the local Bluetooth adapter
        mBtAdapter = BluetoothAdapter.getDefaultAdapter();
        if (mBtAdapter != null)
                doDiscovery();
}

@Override
protected void onDestroy()
{
        super.onDestroy();

        // Make sure we're not doing discovery anymore
        if (mBtAdapter != null)
        {
                mBtAdapter.cancelDiscovery();
        }

        // Unregister broadcast listeners
        this.unregisterReceiver(mReceiver);
}

/**
 * Start device discover with the BluetoothAdapter
 */
private void doDiscovery()
{
        if (D) Log.d(TAG, "doDiscovery()");

        // Indicate scanning in the title
        //setProgressBarIndeterminateVisibility(true);
        setTitle(R.string.scanning);

        // If we're already discovering, stop it
        if (mBtAdapter.isDiscovering())
        {
                mBtAdapter.cancelDiscovery();
        }
        hasDevices = false;
        // Request discover from BluetoothAdapter
        mBtAdapter.startDiscovery();
        mpDialog = new ProgressDialog(DeviceListActivity.this);
mpDialog.setProgressStyle(ProgressDialog.STYLE_SPINNER);//设置风格为圆形进度条
mpDialog.setTitle("Remind");//设置标题
mpDialog.setMessage("Scaning the bluetooth devices...");
mpDialog.setIndeterminate(false);//设置进度条是否为不明确
mpDialog.setCancelable(true);//设置进度条是否可以按退回键取消
mpDialog.setButton("Stop", new DialogInterface.OnClickListener(){

    @Override
    public void onClick(DialogInterface dialog, int which) {
        dialog.cancel();
        if (mBtAdapter.isDiscovering()){
                mBtAdapter.cancelDiscovery();
        }
    }
});
mpDialog.show();
}

// The on-click listener for all devices in the ListViews
private OnItemClickListener mDeviceClickListener = new OnItemClickListener()
{
        public void onItemClick(AdapterView<?> av, View v, int arg2, long arg3)
```

```
        {
                        // Cancel discovery because it's costly and we're about to connect
                mBtAdapter.cancelDiscovery();

                // Get the device MAC address, which is the last 17 chars in the
                // View
                if (hasDevices){
                        String info = ((TextView) v).getText().toString();
                        String address = info.substring(info.length() - 17);

                        // Create the result Intent and include the MAC address
                        Intent intent = new Intent();
                        intent.putExtra(EXTRA_DEVICE_ADDRESS, address);

                        // Set result and finish this Activity
                        setResult(Activity.RESULT_OK, intent);
                        finish();
                }
        }
};


// The BroadcastReceiver that listens for discovered devices and
// changes the title when discovery is finished
private final BroadcastReceiver mReceiver = new BroadcastReceiver()
{
        @Override
        public void onReceive(Context context, Intent intent)
        {
                String action = intent.getAction();
                // When discovery finds a device
                if (BluetoothDevice.ACTION_FOUND.equals(action))
                {
                        // Get the BluetoothDevice object from the Intent
                        BluetoothDevice device = (BluetoothDevice)intent
                                        .getParcelableExtra(BluetoothDevice.EXTRA_DEVICE);
                        String tempString;
                        if(device.getBondState() == BluetoothDevice.BOND_NONE){
                                tempString = "Status: UnPaired\n";
                        }
                        else {
                                tempString = "Status: Paired\n";
                        }

                        //添加设备
                        tempString += device.getName() + "\n"
                                        + device.getAddress();
                        //防止重复添加
                        if (lstDevices.indexOf(tempString) == -1){
                                lstDevices.add(tempString);
                                mNewDevicesArrayAdapter.notifyDataSetChanged();
                        }
                        //mNewDevicesArrayAdapter.add(device.getName() + "\n"
                        //              + device.getAddress());
                        hasDevices = true;

                }
                else if (BluetoothAdapter.ACTION_DISCOVERY_FINISHED.equals(action))
                {
                        // When discovery is finished, change the Activity title
                        //setProgressBarIndeterminateVisibility(false);
                        setTitle(R.string.select_device);
                        if (mNewDevicesArrayAdapter.getCount() == 0)
                        {
```

```
                String noDevices = getResources().getText(
                        R.string.none_found).toString();
                mNewDevicesArrayAdapter.add(noDevices);
                hasDevices = false;
            }
            mpDialog.cancel();
        }
    }
};

}
```

7：点击listview中蓝牙设备条目，返回前面的onAtivityResult中，此刻，我手机获取到了服务端的address,主动请求服务端

```
/**
 * 连接蓝牙设备
 */
public void ConnectDevices(final String address){
        // Get the BLuetoothDevice object
BluetoothDevice device = mBluetoothAdapter.getRemoteDevice(address);  //根据前面获取的address,获取一个BlueToothDevices[代表这
// Attempt to connect to the device
mBtService.connect(device);
}
```

8：进行蓝牙连接

```
/**
    * This thread runs while attempting to make an outgoing connection
    * with a device. It runs straight through; the connection either
    * succeeds or fails.
    */
   private class ConnectThread extends Thread {
       private final BluetoothSocket mmSocket;
       private final BluetoothDevice mmDevice;

       public ConnectThread(BluetoothDevice device) {
           mmDevice = device;
           BluetoothSocket tmp = null;

           // Get a BluetoothSocket for a connection with the
           // given BluetoothDevice
           try {
               int sdk = Build.VERSION.SDK_INT;
               if(sdk >= 10){
                       tmp = device.createInsecureRfcommSocketToServiceRecord(MY_UUID);
               }else {
                       tmp = device.createRfcommSocketToServiceRecord(MY_UUID);
                               }
           } catch (IOException e) {
               Log.e(TAG, "create() failed", e);
           }
           mmSocket = tmp;
       }

       public void run() {
           Log.i(TAG, "BEGIN mConnectThread");
           setName("ConnectThread");

           // Always cancel discovery because it will slow down a connection
```

```
            mAdapter.cancelDiscovery();

            // Make a connection to the BluetoothSocket
            try {
                // This is a blocking call and will only return on a
                // successful connection or an exception
                mmSocket.connect();
            } catch (IOException e) {
                connectionFailed();
                // Close the socket
                try {
                    mmSocket.close();
                } catch (IOException e2) {
                    Log.e(TAG, "unable to close() socket during connection failure", e2);
                }
                // Start the service over to restart listening mode
                BluetoothService.this.start();
                return;
            }

            // Reset the ConnectThread because we're done
            synchronized (BluetoothService.this) {
                mConnectThread = null;
            }

            // Start the connected thread
            connected(mmSocket, mmDevice);
        }

        public void cancel() {
            try {
                mmSocket.close();
            } catch (IOException e) {
                Log.e(TAG, "close() of connect socket failed", e);
            }
        }
    }
```

9：连接成功后即可进行数据读取

```
/**
     * This thread runs during a connection with a remote device.
     * It handles all incoming and outgoing transmissions.
     */
    private class ConnectedThread extends Thread {
        private final BluetoothSocket mmSocket;
        private final InputStream mmInStream;
        private final OutputStream mmOutStream;

        public ConnectedThread(BluetoothSocket socket) {
            Log.d(TAG, "create ConnectedThread");
            mmSocket = socket;
            InputStream tmpIn = null;
            OutputStream tmpOut = null;

            // Get the BluetoothSocket input and output streams
            try {
                tmpIn = socket.getInputStream();
                tmpOut = socket.getOutputStream();
            } catch (IOException e) {
                Log.e(TAG, "temp sockets not created", e);
            }

            mmInStream = tmpIn;
```

```
            mmOutStream = tmpOut;
        }

        public void run() {
            Log.i(TAG, "BEGIN mConnectedThread");
            byte[] buffer = new byte[1024];
            int bytes;

            // Keep listening to the InputStream while connected
            receiveBuffer = "";
            while (true) {
                try {
                        while(true){
                                // Read from the InputStream
                                        bytes = mmInStream.read(buffer);
                                        //Log.i(TAG, "收到串口数据: " + (new String(buffer,0,bytes)) + "(" + String.valueOf(bytes)

                                        String tempString = new String(buffer,0,bytes);
                                        receiveBuffer += tempString;

                                        if (receiveBuffer.endsWith("\r\n")){
                                                String strArray[] = receiveBuffer.split("\r\n");
                                                for(String stemp:strArray){
                                                        //发送显示
                                                        mHandler.obtainMessage(BluetoothSet.MESSAGE_READ, stemp.length(), -1, stem
                                                        .sendToTarget();
                                                        sleep(10L);
                                                }
                                                //初始化接收缓存数据
                                                receiveBuffer = "";
                                        }

                                        if (mmInStream.available() == 0) break;
                        }
                } catch (IOException e) {
                    Log.e(TAG, "disconnected", e);
                    connectionLost();
                    break;
                } catch (InterruptedException e) {
                        Log.e(TAG, "disconnected", e);
                    connectionLost();
                    break;
                                }
            }
        }

        /**
         * Write to the connected OutStream.
         * @param buffer  The bytes to write
         */
        public void write(byte[] buffer) {
                try {
                        mmOutStream.write(buffer);

                        // Share the sent message back to the UI Activity
                    mHandler.obtainMessage(BluetoothSet.MESSAGE_WRITE, -1, -1, buffer)
                            .sendToTarget();

                }catch (IOException e) {
                                Log.e(TAG, "Exception during write", e);
                        }
        }

        public void cancel() {
```

```
            try {
                                mmSocket.close();
            } catch (IOException e) {
                Log.e(TAG, "close() of connect socket failed", e);
            }
        }
    }
```