宝贵建议请发送至：wangzhenyang@itcast.cn

黑马程序员 itheima.com -编程，始于黑马

# Android 课程同步笔记

Alpha 0.01 版

By 阳哥

# Android 手机卫士-06

## 1. 通讯卫士（★★★）

在本节中我们将完成通讯卫士功能的开发。在功能列表页面点击通讯卫士，则进入黑名单管理界面，界面效果如图。


黑名单管理的原理，首先选择管理按钮，弹出自定义对话框，在对话框中让用户输入要拦截的号码，并且选择要拦截的类型，拦截的类型分为短信拦截、电话拦截、全部拦截。当用户确定后，那么会将这些要拦截的电话以及拦截类型保存在本地数据库中。同时我们会编写一个 Service 服务，在该服务里面我们监听所有的短信和电话，当有短信来时我们拿到短信发送者的号码，如果改号码在黑马单里面则要么拦截期短信（终止广播或者删除短信数据库），要么拦截其电话（挂断电话）。

步骤：

1、编写通讯卫士的布局文件 callsms_safe_activity.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >
    <RelativeLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:background="#8866ff00" >
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="55dp"
```

```
            android:gravity="center"
            android:text="黑名单管理"
            android:textColor="#000000"
            android:textSize="20sp" />

        <Button
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_alignParentRight="true"
            android:onClick="addBlackNumber"
            android:text="管理" />
    </RelativeLayout>

    <FrameLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent" >

        <ListView
            android:id="@+id/list_callsms_safe"
            android:layout_width="match_parent"
            android:layout_height="wrap_content" >
        </ListView>
        <LinearLayout
            android:id="@+id/ll_logging"
            android:layout_height="match_parent"
            android:layout_width="match_parent"
            android:gravity="center"
            android:orientation="vertical"
            >
        <ProgressBar
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            />
        <TextView
            android:layout_height="wrap_content"
            android:layout_width="wrap_content"
            android:text="数据正在加载中..."
            />
        </LinearLayout>

    </FrameLayout>
</LinearLayout>
```

2、我们把黑名单信息封装成 BlackNumberInfo 对象。

```
public class BlackNumberInfo {
    private String number;
    private String model;
    //省略 setter&getter
```

3、创建 BlackNumberDao 类用于提供操作黑名单接口。

```
public class BlackNumberDao {
    private BlackNumberOpenHelper openHelper;
    public BlackNumberDao(Context context){
        openHelper = new BlackNumberOpenHelper(context);
    }
    /**
     *
     * @param number
     * @param model
     */
    public void add(String number,String model){
        SQLiteDatabase database = openHelper.getWritableDatabase();
        ContentValues values = new ContentValues();
        values.put("number", number);
        values.put("model", model);
        database.insert("blacknumber", null, values );
        database.close();
    }
    public void delete(String number){
        SQLiteDatabase database = openHelper.getWritableDatabase();
        database.delete("blacknumber", "number=?", new String[]{number});
        database.close();
    }
    public void update(String number,String model){
        SQLiteDatabase database = openHelper.getWritableDatabase();
        ContentValues values = new ContentValues();
        values.put("model", model);
        database.update("blacknumber", values, "number=?", new String[]{number});
        database.close();
    }
    public boolean find(String number){
        SQLiteDatabase database = openHelper.getReadableDatabase();
           Cursor query = database.query("blacknumber", null,"number=?" , new
String[]{number}, null, null, null);
```

```
        if (query.moveToNext()) {
            database.close();
            return true;
        }else {
            database.close();
            return false;
        }
    }
    public String findModel(String number){
        SQLiteDatabase database = openHelper.getReadableDatabase();
        Cursor query = database.query("blacknumber", new
String[]{"model"},"number=?" , new String[]{number}, null, null, null);
        String model = null;
        while(query.moveToNext()){
            model = query.getString(0);
        }
        database.close();
        return model;

    }
    public List<BlackNumberInfo> findAll(){
        SystemClock.sleep(2000);
        SQLiteDatabase database = openHelper.getReadableDatabase();
        List<BlackNumberInfo> list = new ArrayList<BlackNumberInfo>();
        Cursor cursor = database.query("blacknumber", new String[]{"number","model"},
null, null, null, null, null);
        while(cursor.moveToNext()){
            String number = cursor.getString(0);
            String model = cursor.getString(1);
            BlackNumberInfo info = new BlackNumberInfo(number,model);
            list.add(info);
        }
        cursor.close();
        database.close();
        return list;
    }
    public List<BlackNumberInfo> findPart(int index){
//      SystemClock.sleep(2000);
        SQLiteDatabase database = openHelper.getReadableDatabase();
        List<BlackNumberInfo> list = new ArrayList<BlackNumberInfo>();
        String sql = "select number,model from blacknumber order by _id desc limit 20
offset ? ";
            Cursor cursor = database.rawQuery(sql , new
```

```
String[]{index+""});//("blacknumber", new String[]{"number","model"}, null, null,
null, null, null);
        while(cursor.moveToNext()){
            String number = cursor.getString(0);
            String model = cursor.getString(1);
            BlackNumberInfo info = new BlackNumberInfo(number,model);
            list.add(info);
        }
        cursor.close();
        database.close();
        return list;
    }
    public int getTotal(){
        int total = 0;
        SQLiteDatabase database = openHelper.getReadableDatabase();
        Cursor cursor = database.rawQuery("select count(*) from blacknumber ", null);
        if (cursor.moveToNext()) {
            total = cursor.getInt(0);
        }
        cursor.close();
        database.close();
        return total;
    }
    }
```

4、创建 CallSMSSafeActivity 类，在该类中进行核心业务的操作。

```
public class CallSMSSafeActivity extends Activity {
    // 声明 ListView 对象
    private ListView list_callsms_safe;
    // dao 对象
    private BlackNumberDao dao;
    // 声明集合存储黑名单
    private List<BlackNumberInfo> blackNumberInfos;
    // 声明 LinearLayout 对象
    private LinearLayout ll_logging;
    private int index = 0;
    private Integer total = null;
    // 声明一个 ListView 的适配器
    private MyAdapter myAdapter;
    private boolean isLoading = false;
```

```
    Handler handler = new Handler() {
        public void handleMessage(android.os.Message msg) {
            if (msg.what == RESULT_OK) {
                ll_logging.setVisibility(View.INVISIBLE);
                if (myAdapter == null) {
                    // 如果适配器为 null，则新建
                    myAdapter = new MyAdapter();
                    list_callsms_safe.setAdapter(myAdapter);
                } else {
                    // 通知适配器更新数据
                    myAdapter.notifyDataSetChanged();
                }
                isLoading = false;
            }
        };
    };

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

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.callsms_safe_activity);
        list_callsms_safe = (ListView) findViewById(R.id.list_callsms_safe);
        // 初始化数据库对象
        dao = new BlackNumberDao(this);
        // 获取线性布局中的线性布局控件
        ll_logging = (LinearLayout) findViewById(R.id.ll_logging);
        // 开启一个子线程，从数据库中加载数据
        fillData();
        /*
         * 分页加载数据的效果 给 ListView 添加滚动监听器 当滚动到最下面的时候加载下一页数据，直到加载完所有数据为止
         */
        list_callsms_safe.setOnScrollListener(new OnScrollListener() {
            @Override
            public void onScrollStateChanged(AbsListView view, int scrollState) {
                switch (scrollState) {
                case OnScrollListener.SCROLL_STATE_IDLE:// 停止状态
                    //获取最后一个可见的脚标
```

```
                int visiblePosition = view.getLastVisiblePosition();
                //如果最后一个脚标正好等于所有数据个数-1，也就是数据正好的欧显示完
                if (visiblePosition == blackNumberInfos.size() - 1) {
                    if (isLoading) {
                        Toast.makeText(CallSMSSafeActivity.this, "不要着急，数据正在
加载中", 0).show();
                        return;
                    }
                    Toast.makeText(CallSMSSafeActivity.this, "加载更多数据",
0).show();
                    //如果 index 大于所有数据则提示没有数据
                    if (index >= getTotal()) {
                        Toast.makeText(CallSMSSafeActivity.this, "已经没有数据了",
Toast.LENGTH_SHORT).show();
                        return;
                    }
                    //如果 index+20 大于所有则计算最大的 index
                    if (index + 20 >= getTotal()) {
                        index = blackNumberInfos.size() + (index + 20 - getTotal());
                    } else {
                        //否则 index+20 即可
                        index += 20;
                    }

                    System.out.println("index=" + index);
                    fillData();
                } else if (view.getFirstVisiblePosition() == 0) {
                    // index-=blackNumberInfos.size();
                    // fillData();
                }
                break;
            case OnScrollListener.SCROLL_STATE_FLING:// 滑动状态
                break;

            default:
                break;
            }
        }

        @Override
        public void onScroll(AbsListView view, int firstVisibleItem, int
visibleItemCount, int totalItemCount) {
```

```
            }
        });
    }
    //在子线程中加载数据，因为访问数据库属于耗时操作
    private void fillData() {
        //加载数据的时候提示正在加载
        ll_logging.setVisibility(View.VISIBLE);
        new Thread(new Runnable() {

            @Override
            public void run() {
                if (blackNumberInfos == null) {
                    //第一次加载
                    blackNumberInfos = dao.findPart(index);
                } else {
                    //分页加载时将每一页数据都添加到集合中
                    blackNumberInfos.addAll(dao.findPart(index));
                }
                //发送消息给 ListView 的适配器告诉其数据已经加载完毕
                handler.sendEmptyMessage(RESULT_OK);
            }
        }).start();
    }
    //从数据库中获取总数据量
    private int getTotal() {
        if (total == null) {
            total = dao.getTotal();
        }
        return total;
    }
    //声明一个对话框
    private AlertDialog alertDialog;
    //添加黑名单
    public void addBlackNumber(View view) {
        Toast.makeText(this, "添加黑马名单", 0).show();
        //创建一个对话框对象
        alertDialog = new Builder(this).create();
        //用打气筒将一个布局填充为 View 对象
        View viewContext = View.inflate(this, R.layout.dialog_add_blacknumber, null);
        alertDialog.setView(viewContext, 0, 0, 0, 0);
        alertDialog.show();
        final EditText et_number = (EditText)
viewContext.findViewById(R.id.et_number);
```

```
        final RadioGroup rg_model = (RadioGroup)
viewContext.findViewById(R.id.rg_model);
        Button bn_ok = (Button) viewContext.findViewById(R.id.bn_ok);
        Button bn_cancel = (Button) viewContext.findViewById(R.id.bn_cancel);
        bn_cancel.setOnClickListener(new OnClickListener() {

            @Override
            public void onClick(View v) {
                alertDialog.dismiss();
            }
        });
        bn_ok.setOnClickListener(new OnClickListener() {

            @Override
            public void onClick(View v) {
                // 获取电话号码
                String number = et_number.getText().toString();
                if (TextUtils.isEmpty(number) || number.trim().length() < 3) {
                    Toast.makeText(CallSMSSafeActivity.this, "请输入电话号码！",
0).show();
                    return;
                }
                // 获取拦截模式
                int checkedRadioButtonId = rg_model.getCheckedRadioButtonId();
                String model = null;
                switch (checkedRadioButtonId) {
                case R.id.rb_all:// 全部拦截
                    model = "2";
                    break;
                case R.id.rb_phone:// 拦截电话
                    model = "0";
                    break;
                case R.id.rb_sms:// 拦截短信
                    model = "1";
                    break;
                default:
                    Toast.makeText(CallSMSSafeActivity.this, "请设置拦截模式",
Toast.LENGTH_SHORT).show();
                    return;
                }
                // 保存到数据库
                BlackNumberInfo info = new BlackNumberInfo(number, model);
                blackNumberInfos.add(0, info);
```

```
                dao.add(number, model);
                myAdapter.notifyDataSetChanged();
                alertDialog.dismiss();
            }
        });
    }
    //自定义 ListView 适配器继承 BaseAdapter
    class MyAdapter extends BaseAdapter {

        @Override
        public int getCount() {
            return blackNumberInfos.size();
        }
        @Override
        public Object getItem(int position) {
            return null;
        }
        @Override
        public long getItemId(int position) {
            return 0;
        }
        /**
         * 获取 view 对象，这里面对 ListView 进行了优化
         */
        @Override
        public View getView(final int position, View convertView, ViewGroup parent) {
            ViewHoler viewHoler;
            View view;
            if (convertView != null) {
                //convertView 是缓存的 View 对象，如果不为空则可以直接使用
                view = convertView;
                viewHoler = (ViewHoler) view.getTag();
            } else {
                viewHoler = new ViewHoler();
                //用打气筒填充一个 View 对象
                view = View.inflate(CallSMSSafeActivity.this,
R.layout.list_callsms_item, null);
                //从填充好的 View 对象中获取里面包含的控件对象，并赋值给我们自定义的类
ViewHolder
                viewHoler.tv_number = (TextView) view.findViewById(R.id.tv_number);
                viewHoler.tv_model = (TextView) view.findViewById(R.id.tv_model);
                viewHoler.iv_delete = (ImageView) view.findViewById(R.id.iv_delete);
                //将 ViewHolder 对象绑定给 View 对象上
```

```
            view.setTag(viewHoler);
        }
        final BlackNumberInfo blackNumberInfo = blackNumberInfos.get(position);
        viewHoler.tv_number.setText(blackNumberInfo.getNumber());
        //判断拦截类型
        if (blackNumberInfo.getModel().equals("0")) {
            viewHoler.tv_model.setText("电话拦截");
        } else if (blackNumberInfo.getModel().equals("1")) {
            viewHoler.tv_model.setText("短信拦截");
        } else if (blackNumberInfo.getModel().equals("2")) {
            viewHoler.tv_model.setText("电话+短信拦截");
        }
        //给删除图标绑定点击事件，实现点击后删除功能
        viewHoler.iv_delete.setOnClickListener(new OnClickListener() {

            @Override
            public void onClick(View v) {
                dao.delete(blackNumberInfo.getNumber());
                blackNumberInfos.remove(position);
                myAdapter.notifyDataSetChanged();
            }
        });
        return view;
    }
}
/*
 * 定义一个 ViewHolder 类
 * 该类中保存三个属性都是 ListView 中每个条目的属性，这样就不用每次都从布局文件中查询控
件
 * 这也是 ListView 优化的一个重要方面
 */
class ViewHoler {
    TextView tv_number;
    TextView tv_model;
    ImageView iv_delete;
}
}
```

5、在上面的代码中我们用到了 ListView，ListView 中的条目的布局文件 list_callsms_item.xml 清单如下：

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content" >
    <TextView
        android:textColor="#000000"
        android:textSize="18sp"
        android:id="@+id/tv_number"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="电话"
        />
    <TextView
        android:layout_below="@id/tv_number"
        android:textColor="#88000000"
        android:textSize="16sp"
        android:id="@+id/tv_model"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="model"
        />
    <ImageView
        android:layout_centerVertical="true"
        android:layout_alignParentRight="true"
        android:id="@+id/iv_delete"
        android:layout_height="50dp"
        android:layout_width="50dp"
        android:src="@drawable/button_delete"
        />

</RelativeLayout>
```

6、在第 4 步骤中我们用到了自定义的 AlertDialog 对象，其对应的布局文件 dialog_add_blacknumber.xml：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="300dp"
    android:layout_height="wrap_content"
    android:background="#ffffff"
    android:gravity="center"
    android:orientation="vertical" >
    <TextView
        android:layout_width="match_parent"
```

```
        android:layout_height="50dp"
        android:background="#66ff6600"
        android:gravity="center"
        android:text="添加黑名单"
        android:textSize="20sp" />
    <EditText
        android:id="@+id/et_number"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="请输入要拦截的号码"
        android:inputType="phone" />

    <RadioGroup
        android:id="@+id/rg_model"
        android:orientation="horizontal"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" >
        <RadioButton android:id="@+id/rb_sms" android:layout_width="wrap_content"
android:layout_height="wrap_content" android:text="拦截短信"/>
        <RadioButton android:id="@+id/rb_phone"
android:layout_height="wrap_content" android:text="拦截电话"/>
        <RadioButton android:id="@+id/rb_all" android:layout_width="wrap_content"
android:layout_height="wrap_content" android:text="全部拦截"/>
    </RadioGroup>
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:gravity="center"
        android:orientation="horizontal" >
        <Button
            android:id="@+id/bn_cancel"
            android:layout_width="140dp"
            android:layout_height="wrap_content"
            android:text="取消" />
        <Button
            android:id="@+id/bn_ok"
            android:layout_width="140dp"
            android:layout_height="wrap_content"
            android:text="确定" />
    </LinearLayout>
</LinearLayout>
```

# 2. 短信拦截、电话拦截的实现（★★★）

在第一节我们实现了用户黑名单管理，只是将黑名单添加进了数据库，那么接下来我们将实现开启服务，在服务里面监听短信、监听电话。当拦截到的短信属于黑名单的时候就终止广播或者从数据库中删除短信，当电话属于黑名单的时候我们用代码挂断电话。

步骤：

1、编写 CallSMSSafeService 集成 Service 类，同时在 AndroidManifest.xml 中进行注册。

2、CallSMSSafeService 代码清单如下，对于挂断电话需要用到一个知识点，aidl。关于 aidl 我们会在下面一节详细介绍，这里先给出代码。

```
public class CallSMSSafeService extends Service {
    //自定义的 BroadcastReceiver 类
    private InnerReceiver receiver;
    private BlackNumberDao dao;
    private TelephonyManager tm;
    @Override
    public IBinder onBind(Intent intent) {
        return null;
    }
    @Override
    public void onCreate() {
        super.onCreate();
        //初始化 InnerReceiver 类
        receiver = new InnerReceiver();
        dao = new BlackNumberDao(this);
        //注册短信接收者
        IntentFilter filter = new IntentFilter();
        filter.addAction("android.provider.Telephony.SMS_RECEIVED");
        filter.setPriority(1000);
        registerReceiver(receiver, filter);
        //获取 TelephonyManager 对象
        tm = (TelephonyManager) getSystemService(TELEPHONY_SERVICE);
        //给 TelephonyManager 对象注册拨打电话状态监听
        tm.listen(new MyPhoneStateListener(), PhoneStateListener.LISTEN_CALL_STATE);
    }
```

```
    @Override
    public void onDestroy() {
        super.onDestroy();
        //取消监听
        unregisterReceiver(receiver);
        tm.listen(null, PhoneStateListener.LISTEN_NONE);
    }
    //定义一个类，继承 PhoneStateListener 用于监听电话状态
    class MyPhoneStateListener extends PhoneStateListener {

        @Override
        public void onCallStateChanged(int state, String incomingNumber) {
            super.onCallStateChanged(state, incomingNumber);
            switch (state) {
            case TelephonyManager.CALL_STATE_RINGING:// 电话打进来
                String model = dao.findModel(incomingNumber);
                Toast.makeText(CallSMSSafeService.this, "电话拦截到了=" +
incomingNumber + "/model=" + model, Toast.LENGTH_SHORT).show();
                if ("2".equals(model) || "0".equals(model)) {
                    //挂断电话
                    endCall();
                    //直接删除电话记录
                    deleteCallLog(incomingNumber);
                    //通过 ContentObserver 删除数据库中的记录，这种方法是应该选择的
                    Uri url = Uri.parse("content://call_log/calls");
                    getContentResolver().registerContentObserver(url, true, new
MyObserver(new Handler(), incomingNumber));
                }
                break;

            default:
                break;
            }
        }
    }
    private void endCall() {
        try {
            //通过反射调用 TelephonyManager 的 getITelephony 方法，然后调用 endCall 方法挂
断电话
            Method method = TelephonyManager.class.getDeclaredMethod("getITelephony",
null);
            method.setAccessible(true);
            ITelephony iTelephony = (ITelephony) method.invoke(tm, null);
```

```
    Toast.makeText(this, "拦截掉了一个电话", Toast.LENGTH_SHORT).show();
        } catch (Exception e) {
            System.err.println(e);
        }
    }
    private void deleteCallLog(String number) {
        ContentResolver resolver = getContentResolver();
        Uri url = Uri.parse("content://call_log/calls");
        resolver.delete(url, "number=?", new String[] { number });
    }
    //定义一个类监听数据库变化
    class MyObserver extends ContentObserver {
        private String number;

        public MyObserver(Handler handler, String number) {
            super(handler);
            this.number = number;
        }

        @Override
        public void onChange(boolean selfChange) {
            super.onChange(selfChange);
            //取消数据库监听
            getContentResolver().unregisterContentObserver(this);
            //删除来电号码记录
            deleteCallLog(number);
        }
    }
    // 定义一个短信接收的 BroadcastReceiver 类
    class InnerReceiver extends BroadcastReceiver {

        @Override
        public void onReceive(Context context, Intent intent) {
            // android.provider.Telephony.SMS_RECEIVED
            Object[] objs = (Object[]) intent.getExtras().get("pdus");
            for (Object obj : objs) {
                SmsMessage pdu = SmsMessage.createFromPdu((byte[]) obj);
                String address = pdu.getOriginatingAddress();
                String model = dao.findModel(address);
                if ("1".equals(model) || "2".equals(model)) {
                    System.out.println("拦截到一条短信: " + pdu.getOriginatingAddress()
+ "=" + pdu.getMessageBody());
                    abortBroadcast();
```

```
            }
        }
    }
}
```

# 3. Android 中的 AIDL（★★★）

在上一节中我们用通过反射调用了挂断电话功能，这里面用到了 Android 的 AIDL 技术。因此下面将详细讲解一下 AIDL 的用法。

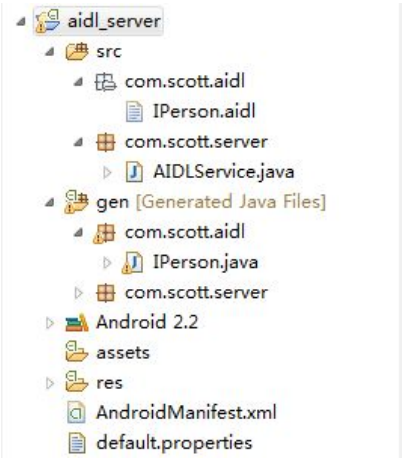
## 3.1 什么是 AIDL

AIDL 是 Android Interface definition language 的缩写，一看就明白，它是一种 Android 内部进程通信接口的描述语言，通过它我们可以定义进程间的通信接口。

## 3.2 AIDL 的使用方式

下面通过一个例子来演示 AIDL 的用法。

步骤：1、我们需要建立一个服务端的工程，如图所以：




2、在 IPerson.aidl 中我们定义了一个“问候”的方法，代码如下：

```
package com.scott.aidl;
interface IPerson {
    String greet(String someone);
}
```

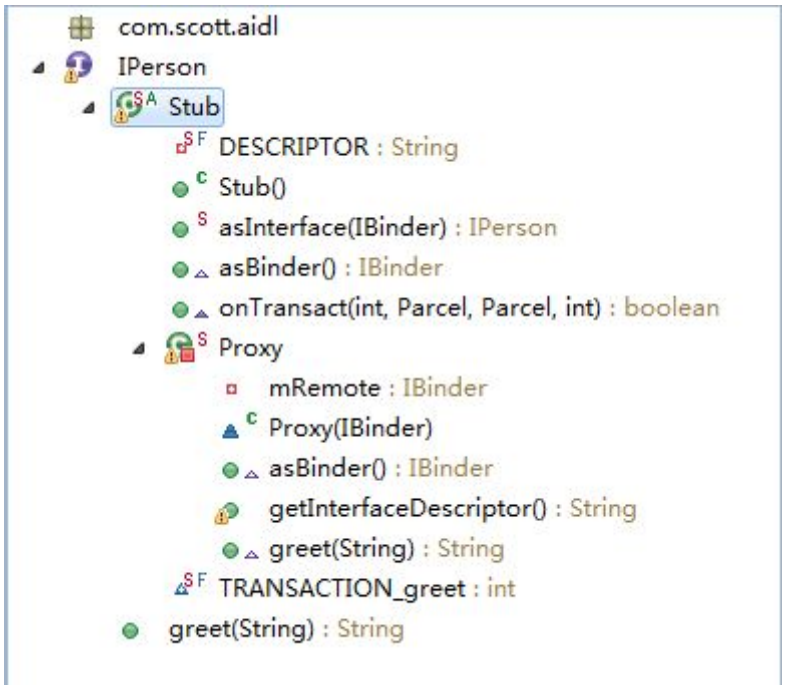
在 Eclipse 插件的帮助下，编译器会自动在 gen 目录中生成对应的 IPerson.java 文件，格式化后的代码如下：

```
package com.scott.aidl;
public interface IPerson extends android.os.IInterface {
    /** Local-side IPC implementation stub class. */
    public static abstract class Stub extends android.os.Binder implements
com.scott.aidl.IPerson {

        private static final java.lang.String DESCRIPTOR = "com.scott.aidl.IPerson";

        /** Construct the stub at attach it to the interface. */
        public Stub() {
            this.attachInterface(this, DESCRIPTOR);
        }

        /**
         * Cast an IBinder object into an com.scott.aidl.IPerson interface,
         * generating a proxy if needed.
         */
        public static com.scott.aidl.IPerson asInterface(android.os.IBinder obj) {
            if ((obj == null)) {
                return null;
            }
            android.os.IInterface iin = (android.os.IInterface)
obj.queryLocalInterface(DESCRIPTOR);
            if (((iin != null) && (iin instanceof com.scott.aidl.IPerson))) {
                return ((com.scott.aidl.IPerson) iin);
            }
            return new com.scott.aidl.IPerson.Stub.Proxy(obj);
        }
          public android.os.IBinder asBinder() {
            return this;
        }
          @Override
        public boolean onTransact(int code, android.os.Parcel data, android.os.Parcel
reply, int flags)
                throws android.os.RemoteException {
            switch (code) {
```

```
            case INTERFACE_TRANSACTION: {
                reply.writeString(DESCRIPTOR);
                return true;
            }
            case TRANSACTION_greet: {
                data.enforceInterface(DESCRIPTOR);
                java.lang.String _arg0;
                _arg0 = data.readString();
                java.lang.String _result = this.greet(_arg0);
                reply.writeNoException();
                reply.writeString(_result);
                return true;
            }
            }
            return super.onTransact(code, data, reply, flags);
        }

        private static class Proxy implements com.scott.aidl.IPerson {
            private android.os.IBinder mRemote;

            Proxy(android.os.IBinder remote) {
                mRemote = remote;
            }

            public android.os.IBinder asBinder() {
                return mRemote;
            }

            public java.lang.String getInterfaceDescriptor() {
                return DESCRIPTOR;
            }

            public java.lang.String greet(java.lang.String someone) throws
android.os.RemoteException {
                android.os.Parcel _data = android.os.Parcel.obtain();
                android.os.Parcel _reply = android.os.Parcel.obtain();
                java.lang.String _result;
                try {
                    _data.writeInterfaceToken(DESCRIPTOR);
                    _data.writeString(someone);
                    mRemote.transact(Stub.TRANSACTION_greet, _data, _reply, 0);
                    _reply.readException();
                    _result = _reply.readString();
```

```
                } finally {
                    _reply.recycle();
                    _data.recycle();
                }
                return _result;
            }
        }

        static final int TRANSACTION_greet =
(android.os.IBinder.FIRST_CALL_TRANSACTION + 0);
    }

    public java.lang.String greet(java.lang.String someone) throws
android.os.RemoteException;
}
```

该文件的大纲视图如下：




IPerson 接口中的抽象内部类 Stub 继承 android.os.Binder 类并实现 IPerson 接口，比较重要的方法是 asInterface(IBinder)方法，该方法会将 IBinder 类型的对象转换成 IPerson 类型，必要的时候生成一个代理对象返回结果。

3、编写 AIDLService 类，代码清单如下。

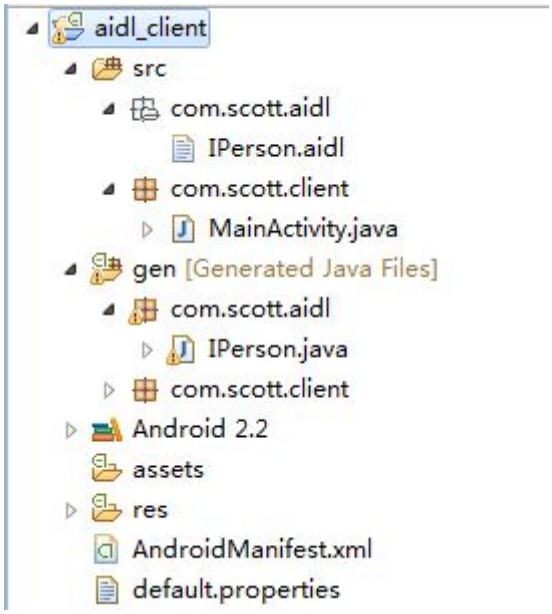
```
package com.scott.server;

import android.app.Service;
import android.content.Intent;
import android.os.IBinder;
import android.os.RemoteException;
import android.util.Log;

import com.scott.aidl.IPerson;

public class AIDLService extends Service {

    private static final String TAG = "AIDLService";

    IPerson.Stub stub = new IPerson.Stub() {
        @Override
        public String greet(String someone) throws RemoteException {
            Log.i(TAG, "greet() called");
            return "hello, " + someone;
        }
    };

    @Override
    public IBinder onBind(Intent intent) {
        Log.i(TAG, "onBind() called");
        return stub;
    }

    @Override
    public boolean onUnbind(Intent intent) {
        Log.i(TAG, "onUnbind() called");
        return true;
    }

    @Override
    public void onDestroy() {
        super.onDestroy();
        Log.i(TAG, "onDestroy() called");
    }
}
```

我们实现了 IPerson.Stub 这个抽象类的 greet 方法，然后再 onBind(Intent)方法中返回我们的 stub 实例，这样一来调用方获取的 IPerson.Stub 就是我们的这个实例，greet 方法也会按照我们的期望那样执行。

4、在 AndroidManifest.xml 注册 AIDLService。

```
<service android:name=".AIDLService" >
          <intent-filter>
              <action android:name="android.intent.action.AIDLService" />
              <category android:name="android.intent.category.DEFAULT" />
          </intent-filter>
 </service>
```

5、服务端已经完成了，再创建一个客户端工程，工程目录结果如下图：




我们只需要把 IPerson.aidl 文件拷到相应的目录中即可，编译器同样会生成相对应的 IPerson.java 文件，这一部分和服务端没什么区别。这样一来，服务端和客户端就在通信协议上达到了统一。我们主要工作在 MainActivity 中完成。

6、MainActivity 代码如下：

```
public class MainActivity extends Activity {
    private Button bindBtn;
    private Button greetBtn;
    private Button unbindBtn;
    private IPerson person;
    private ServiceConnection conn = new ServiceConnection() {
          @Override
        public void onServiceConnected(ComponentName name, IBinder service) {
            Log.i("ServiceConnection", "onServiceConnected() called");
            person = IPerson.Stub.asInterface(service);
        }
```

```
        @Override
        public void onServiceDisconnected(ComponentName name) {
            //This is called when the connection with the service has been unexpectedly
disconnected,
            //that is, its process crashed. Because it is running in our same process,
we should never see this happen.
            Log.i("ServiceConnection", "onServiceDisconnected() called");
        }
    };

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        bindBtn = (Button) findViewById(R.id.bindBtn);
        bindBtn.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent intent = new Intent("android.intent.action.AIDLService");
                bindService(intent, conn, Context.BIND_AUTO_CREATE);

                bindBtn.setEnabled(false);
                greetBtn.setEnabled(true);
                unbindBtn.setEnabled(true);
            }
        });

        greetBtn = (Button) findViewById(R.id.greetBtn);
        greetBtn.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                try {
                    String retVal = person.greet("scott");
                    Toast.makeText(MainActivity.this, retVal,
Toast.LENGTH_SHORT).show();
                } catch (RemoteException e) {
                    Toast.makeText(MainActivity.this, "error",
Toast.LENGTH_SHORT).show();
                }
            }
        });
```

```
        unbindBtn = (Button) findViewById(R.id.unbindBtn);
        unbindBtn.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                unbindService(conn);

                bindBtn.setEnabled(true);
                greetBtn.setEnabled(false);
                unbindBtn.setEnabled(false);
            }
        });
    }
}
```

从代码中可以看到 ,我们要重写 ServiceConnection 中的 onServiceConnected 方法将 IBinder 类型的对像转换成我们的 IPerson 类型。到现在我们就剩下最后一个步骤了，这个环节也是最为关键的，就是绑定我们需要的服务。我们通过服务端 Service 定义的“android.intent.action.AIDLService”这个标识符来绑定其服务，这样客户端和服务端就实现了通信的连接，我们就可以调用 IPerson 中的“问候”方法了。

***至此，本文档完！***

2015 年 1 月 9 日 星期五 00:13:23
北京市海淀区东北旺中路东馨园