第 8 章　丰富你的程序，运用手机多媒体

在过去，手机的功能都比较单调，仅仅就是用来打电话和发短信的。而如今，手机在我们生活中正扮演着越来越重要的角色，各种娱乐方式都可以在手机上进行。上班的路上太无聊，可以带着耳机听音乐。外出旅行的时候，可以在手机上看电影。无论走到哪里，遇到喜欢的事物都可以随手拍下来。

众多的娱乐方式少不了强大的多媒体功能的支持，而 Android 在这一方面也是做得非常出色。它提供了一系列的 API，使得我们可以在程序中调用很多手机的多媒体资源，从而编写出更加丰富多彩的应用程序。本章我们就将对 Android 中一些常用的多媒体功能的使用技巧进行学习。

8.1　使用通知

通知（Notification）是 Android 系统中比较有特色的一个功能，当某个应用程序希望向用户发出一些提示信息，而该应用程序又不在前台运行时，就可以借助通知来实现。发出一条通知后，手机最上方的状态栏中会显示一个通知的图标，下拉状态栏后可以看到通知的详细内容。Android 的通知功能获得了大量用户的认可和喜爱，就连 iOS 系统也在 5.0 版本之后加入了类似的功能。

8.1.1　通知的基本用法

了解了通知的基本概念，下面我们就来看一下通知的使用方法吧。通知的用法还是比较灵活的，既可以在活动里创建，也可以在广播接收器里创建，当然还可以在下一章中我们即将学习的服务里创建。相比于广播接收器和服务，在活动里创建通知的场景还是比较少的，因为一般只有当程序进入到后台的时候我们才需要使用通知。

不过，无论是在哪里创建通知，整体的步骤都是相同的，下面我们就来学习一下创建通知的详细步骤。首先需要一个 NotificationManager 来对通知进行管理，可以调用 Context 的 getSystemService()方法获取到。getSystemService()方法接收一个字符串参数用于确定获取系统的哪个服务，这里我们传入 Context.NOTIFICATION_SERVICE 即可。因此，获取

NotificationManager 的实例就可以写成：

```
NotificationManager manager = (NotificationManager)

getSystemService(Context.NOTIFICATION_SERVICE);
```

接下来需要创建一个 Notification 对象，这个对象用于存储通知所需的各种信息，我们可以使用它的有参构造函数来进行创建。Notification 的有参构造函数接收三个参数，第一个参数用于指定通知的图标，比如项目的 res/drawable 目录下有一张 icon.png 图片，那么这里就可以传入 R.drawable.icon。第二个参数用于指定通知的 ticker 内容，当通知刚被创建的时候，它会在系统的状态栏一闪而过，属于一种瞬时的提示信息。第三个参数用于指定通知被创建的时间，以毫秒为单位，当下拉系统状态栏时，这里指定的时间会显示在相应的通知上。因此，创建一个 Notification 对象就可以写成：

```
Notification notification = new Notification(R.drawable.icon, "This is ticker text",
                                    System.currentTimeMillis());
```

创建好了 Notification 对象后，我们还需要对通知的布局进行设定，这里只需要调用 Notification 的 setLatestEventInfo()方法就可以给通知设置一个标准的布局。这个方法接收四个参数，第一个参数是 Context，这个没什么好解释的。第二个参数用于指定通知的标题内容，下拉系统状态栏就可以看到这部分内容。第三个参数用于指定通知的正文内容，同样下拉系统状态栏就可以看到这部分内容。第四个参数我们暂时还用不到，可以先传入 null。因此，对通知的布局进行设定就可以写成：

```
notification.setLatestEventInfo(context, "This is content title", "This is
content text", null);
```

以上工作都完成之后，只需要调用 NotificationManager 的 notify()方法就可以让通知显示出来了。notify()方法接收两个参数，第一个参数是 id，要保证为每个通知所指定的 id 都是不同的。第二个参数则是 Notification 对象，这里直接将我们刚刚创建好的 Notification 对象传入即可。因此，显示一个通知就可以写成：

```
manager.notify(1, notification);
```

到这里就已经把创建通知的每一个步骤都分析完了，下面就让我们通过一个具体的例子来看一看通知到底是长什么样的。

新建一个 NotificationTest 项目，并修改 activity_main.xml 中的代码，如下所示：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
```

```
    android:orientation="vertical" >

    <Button
        android:id="@+id/send_notice"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Send notice"
        />

</LinearLayout>
```

布局文件非常简单，里面只有一个 Send notice 按钮，用于发出一条通知。接下来修改 MainActivity 中的代码，如下所示：

```
public class MainActivity extends Activity implements OnClickListener {

    private Button sendNotice;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        sendNotice = (Button) findViewById(R.id.send_notice);
        sendNotice.setOnClickListener(this);
    }

    @Override
    public void onClick(View v) {
        switch (v.getId()) {
        case R.id.send_notice:
            NotificationManager manager = (NotificationManager)
getSystemService(NOTIFICATION_SERVICE);
            Notification notification = new Notification(R.drawable.
ic_launcher, "This is ticker text", System.currentTimeMillis());
            notification.setLatestEventInfo(this, "This is content title",
"This is content text", null);
            manager.notify(1, notification);
            break;
        default:
            break;
        }
    }
```

```
}
```

可以看到，我们在 Send notice 按钮的点击事件里面完成了通知的创建工作，创建的过程正如前面所描述的一样。现在就可以来运行一下程序了，点击 Send notice 按钮，就会看到有一条通知在系统状态栏显示出来，如图 8.1 所示。

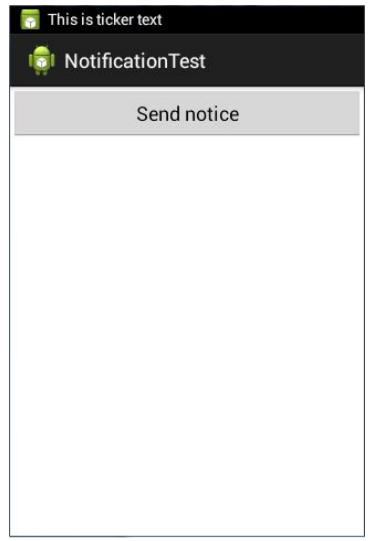

图 8.1

下拉系统状态栏可以看到该通知的详细信息，如图 8.2 所示。

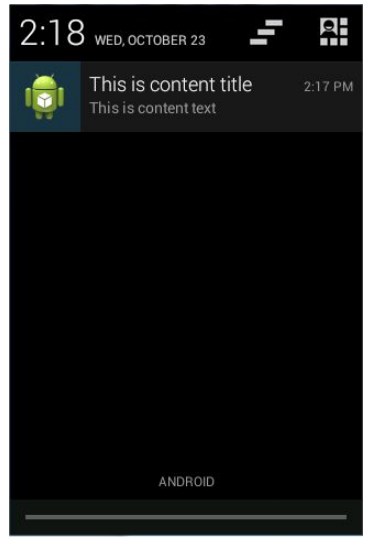

图　8.2

如果你使用过 Android 手机，此时应该会下意识地认为这条通知是可以点击的。但是当你去点击它的时候，你会发现没有任何效果。不对啊，好像每条通知点击之后都应该会有反应的呀？其实要想实现通知的点击效果，我们还需要在代码中进行相应的设置，这就涉及到了一个新的概念，PendingIntent。

PendingIntent 从名字上看起来就和 Intent 有些类似，它们之间也确实存在着不少共同点。比如它们都可以去指明某一个“意图”，都可以用于启动活动、启动服务以及发送广播等。不同的是，Intent 更加倾向于去立即执行某个动作，而 PendingIntent 更加倾向于在某个合适的时机去执行某个动作。所以，也可以把 PendingIntent 简单地理解为延迟执行的 Intent。

PendingIntent 的用法同样很简单，它主要提供了几个静态方法用于获取 PendingIntent 的实例，可以根据需求来选择是使用 getActivity()方法、getBroadcast()方法、还是 getService()方法。这几个方法所接收的参数都是相同的，第一个参数依旧是 Context，不用多做解释。第二个参数一般用不到，通常都是传入 0 即可。第三个参数是一个 Intent 对象，我们可以通过这个对象构建出 PendingIntent 的“意图”。第四个参数用于确定 PendingIntent 的行为，有 FLAG_ONE_SHOT、FLAG_NO_CREATE、FLAG_CANCEL_CURRENT 和 FLAG_UPDATE_CURRENT 这四种值可选，每种值的含义你可以查看文档，我就不一一进行解释了。

对 PendingIntent 有了一定的了解后，我们再回过头来看一下 Notification 的 setLatestEventInfo()方法。刚才我们将 setLatestEventInfo()方法的第四个参数忽略掉了，直接传入了 null，现在仔细观察一下，发现第四个参数正是一个 PendingIntent 对象。因此，这里就可以通过 PendingIntent 构建出一个延迟执行的“意图”，当用户点击这条通知时就会执行相应的逻辑。

现在我们来优化一下 NotificationTest 项目，给刚才的通知加上点击功能，让用户点击它的时候可以启动另一个活动。

首先需要准备好另一个活动，这里新建布局文件 notification_layout.xml，代码如下所示：

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerInParent="true"
        android:textSize="24sp"
        android:text="This is notification layout"
        />
```

```
</RelativeLayout>
```

布局文件的内容非常简单，只有一个居中显示的 TextView，用于展示一段文本信息。然后新建 NotificationActivity 继承自 Activity，在这里加载刚才定义的布局文件，代码如下所示：

```
public class NotificationActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.notification_layout);
    }

}
```

接着修改 AndroidManifest.xml 中的代码，在里面加入 NotificationActivity 的注册声明，如下所示：

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.notificationtest"
    android:versionCode="1"
    android:versionName="1.0" >
    ……
    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        ……
        <activity android:name=".NotificationActivity" >
        </activity>
    </application>

</manifest>
```

这样就把 NotificationActivity 这个活动准备好了，下面我们修改 MainActivity 中的代码，给通知加入点击功能，如下所示：

```
public class MainActivity extends Activity implements OnClickListener {
    ……
    @Override
    public void onClick(View v) {
```

```
        switch (v.getId()) {
        case R.id.send_notice:
            NotificationManager manager = (NotificationManager)
getSystemService(NOTIFICATION_SERVICE);
            Notification notification = new Notification(R.drawable.
ic_launcher, "This is ticker text", System.currentTimeMillis());
            Intent intent = new Intent(this, NotificationActivity.class);
            PendingIntent pi = PendingIntent.getActivity(this, 0, intent,
PendingIntent.FLAG_CANCEL_CURRENT);
            notification.setLatestEventInfo(this, "This is content title",
"This is content text", pi);
            manager.notify(1, notification);
            break;
        default:
            break;
        }
    }

}
```

可以看到，这里先是使用 Intent 表达出我们想要启动 NotificationActivity 的“意图”，然后将构建好的 Intent 对象传入到 PendingIntent 的 getActivity()方法里，以得到 PendingIntent 的实例，接着把它作为第四个参数传入到 Notification 的 setLatestEventInfo()方法中。

现在重新运行一下程序，并点击 Send notice 按钮，依旧会发出一条通知。然后下拉系统状态栏，点击一下该通知，就会看到 NotificationActivity 这个活动的界面了，如图 8.3 所示。

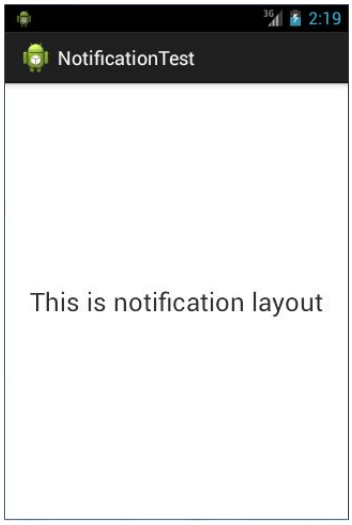

图 8.3

咦？怎么系统状态上的通知图标还没有消失呢？是这样的，如果我们没有在代码中对该通知进行取消，它就会一直显示在系统的状态栏上显示。解决的方法也很简单，调用NotificationManager 的 cancel()方法就可以取消通知了。修改 NotificationActivity 中的代码，如下所示：

```
public class NotificationActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.notification_layout);
        NotificationManager manager = (NotificationManager)
getSystemService(NOTIFICATION_SERVICE);
        manager.cancel(1);
    }

}
```

可以看到，这里我们在 cancel()方法中传入了 1，这个 1 是什么意思呢？还记得在创建通知的时候给每条通知指定的 id 吗？当时我们给这条通知设置的 id 就是 1。因此，如果你想要取消哪一条通知，就在 cancel()方法中传入该通知的 id 就行了。

8.1.2 通知的高级技巧

现在你已经掌握了创建和取消通知的方法，并且知道了如何去响应通知的点击事件。不过通知的用法并不仅仅是这些呢，那么本节中我们就来探究一下通知更多的高级技巧。

观察 Notification 这个类，你会发现里面还有很多我们没有使用过的属性。先来看看 sound 这个属性吧，它可以在通知发出的时候播放一段音频，这样就能够更好地告知用户有通知到来。sound 这个属性是一个 Uri 对象，所以在指定音频文件的时候还需要先获取到音频文件对应的 URI。比如说，我们手机的/system/media/audio/ringtones 目录下有一个 Basic_tone.ogg 音频文件，那么在代码中这样就可以这样指定：

```
    Uri soundUri = Uri.fromFile(new File("/system/media/audio/ringtones/
Basic_tone.ogg"));
    notification.sound = soundUri;
```

除了允许播放音频外，我们还可以在通知到来的时候让手机进行振动，使用的是 vibrate 这个属性。它是一个长整型的数组，用于设置手机静止和振动的时长，以毫秒为单位。下标

为 0 的值表示手机静止的时长，下标为 1 的值表示手机振动的时长，下标为 2 的值又表示手机静止的时长，以此类推。所以，如果想要让手机在通知到来的时候立刻振动 1 秒，然后静止 1 秒，再振动 1 秒，代码就可以写成：

```
long[] vibrates = {0, 1000, 1000, 1000};
notification.vibrate = vibrates;
```

不过，想要控制手机振动还需要声明权限的。因此，我们还得编辑 AndroidManifest.xml 文件，加入如下声明：

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.notificationtest"
    android:versionCode="1"
    android:versionName="1.0" >
    ……
    <uses-permission android:name="android.permission.VIBRATE" />
    ……
</manifest>
```

学会了控制通知的声音和振动，下面我们来看一下如何在通知到来时控制手机 LED 灯的显示。

现在的手机基本上都会前置一个 LED 灯，当有未接电话或未读短信，而此时手机又处于锁屏状态时，LED 灯就会不停地闪烁，提醒用户去查看。我们可以使用 ledARGB、ledOnMS、ledOffMS 以及 flags 这几个属性来实现这种效果。ledARGB 用于控制 LED 灯的颜色，一般有红绿蓝三种颜色可选。ledOnMS 用于指定 LED 灯亮起的时长，以毫秒为单位。ledOffMS 用于指定 LED 灯暗去的时长，也是以毫秒为单位。flags 可用于指定通知的一些行为，其中就包括显示 LED 灯这一选项。所以，当通知到来时，如果想要实现 LED 灯以绿色的灯光一闪一闪的效果，就可以写成：

```
notification.ledARGB = Color.GREEN;
notification.ledOnMS = 1000;
notification.ledOffMS = 1000;
notification.flags = Notification.FLAG_SHOW_LIGHTS;
```

当然，如果你不想进行那么多繁杂的设置，也可以直接使用通知的默认效果，它会根据当前手机的环境来决定播放什么铃声，以及如何振动，写法如下：

```
notification.defaults = Notification.DEFAULT_ALL;
```

注意，以上所涉及的这些高级技巧都要在手机上运行才能看得到效果，模拟器是无法表现出振动、以及 LED 灯闪烁等功能的。

经验值：+10000　　目前经验值：61905

级别：资深鸟

赢得宝物：战胜通知神。拾取通知神掉落的宝物，一只巨大的橡皮鸭子、一把巨大的神界长老赠送的长命锁。因为宝物巨大，无法携带，所以我在当地低价处理了。通知神是体型极其巨大的神兽，即使在身形普遍壮硕的神界也绝对可以当得起“哇！真大”这个词，通知神的外形像短鼻子小耳朵的大象，但比大象大得多得多，其体长可达 30 米，身高超 20 米，重达 200 多吨，尽管其身型巨大，但生性胆小，性情单纯敏感，喜群居，每当受到惊吓，就会拼命用大粗脚跺地以缓解紧张情绪，同时也会以地面的震动通知同伴有可怕的情况发生，而同伴一旦收到震动信号也立马会吓得拼命跺地，结果十几头通知神一起跺地常常让周围数百平方公里的大地震颤不已，地下的穴居小动物们也被震得苦不堪言。神界的巨兽部长老已派人研究如何缓解通知神与生俱来的易受惊吓的性格，并取得了一定成效，现在大家伙们已经不再对偶然飘落到地面的树叶感到害怕。通知神最喜欢的事情是在水里泡澡。

8.2　接收和发送短信

收发短信应该是每个手机最基本的功能之一了，即使是许多年前的老手机也都会具备这项功能，而 Android 作为出色的智能手机操作系统，自然也少不了在这方面的支持。每个 Android 手机都会内置一个短信应用程序，使用它就可以轻松地完成收发短信的操作，如图 8.4 所示。

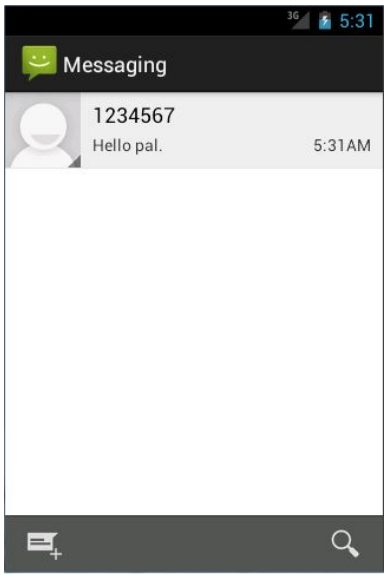

图　8.4

不过作为一名开发者，仅仅满足于此显然是不够的。你要知道，Android 还提供了一系列的 API，使得我们甚至可以在自己的应用程序里接收和发送短信。也就是说，只要你有足够的信心，完全可以自己实现一个短信应用来替换掉 Android 系统自带的短信应用。那么下面我们就来看一看，如何才能在自己的应用程序里接收和发送短信。

8.2.1　接收短信

其实接收短信主要是利用了我们在第 5 章学习过的广播机制。当手机接收到一条短信的时候，系统会发出一条值为 android.provider.Telephony.SMS_RECEIVED 的广播，这条广播里携带着与短信相关的所有数据。每个应用程序都可以在广播接收器里对它进行监听，收到广播时再从中解析出短信的内容即可。

让我们通过一个具体的例子来实践一下吧，新建一个 SMSTest 项目，首先修改 activity_main.xml 中的代码，如下所示：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >

    <LinearLayout
        android:layout_width="match_parent"
```

```
        android:layout_height="50dp" >

        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_gravity="center_vertical"
            android:padding="10dp"
            android:text="From:" />

        <TextView
            android:id="@+id/sender"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_gravity="center_vertical" />
    </LinearLayout>

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="50dp" >

        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_gravity="center_vertical"
            android:padding="10dp"
            android:text="Content:" />

        <TextView
            android:id="@+id/content"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_gravity="center_vertical" />
    </LinearLayout>

</LinearLayout>
```

这个布局文件里，我们在根元素下面放置了两个 LinearLayout，用于显示两行数据。第一个 LinearLayout 中有两个 TextView，用于显示短信的发送方。第二个 LinearLayout 中也有两个 TextView，用于显示短信的内容。

接着修改 MainActivity 中的代码，在 onCreate()方法中获取到两个 TextView 的实例，如

下所示：

```
public class MainActivity extends Activity {

    private TextView sender;

    private TextView content;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        sender = (TextView) findViewById(R.id.sender);
        content = (TextView) findViewById(R.id.content);
    }

}
```

然后我们需要创建一个广播接收器来接收系统发出的短信广播。在 MainActivity 中新建 MessageReceiver 内部类继承自 BroadcastReceiver，并在 onReceive()方法中编写获取短信数据的逻辑，代码如下所示：

```
public class MainActivity extends Activity {

        ……

    class MessageReceiver extends BroadcastReceiver {

        @Override
        public void onReceive(Context context, Intent intent) {
            Bundle bundle = intent.getExtras();
            Object[] pdus = (Object[]) bundle.get("pdus"); // 提取短信消息
            SmsMessage[] messages = new SmsMessage[pdus.length];
            for (int i = 0; i < messages.length; i++) {
                messages[i] = SmsMessage.createFromPdu((byte[]) pdus[i]);
            }
            String address = messages[0].getOriginatingAddress(); // 获取发
送方号码
            String fullMessage = "";
            for (SmsMessage message : messages) {
                fullMessage += message.getMessageBody(); // 获取短信内容
```

```
            }
            sender.setText(address);
            content.setText(fullMessage);
        }

    }
}
```

可以看到，首先我们从 Intent 参数中取出了一个 Bundle 对象，然后使用 pdu 密钥来提取一个 SMS pdus 数组，其中每一个 pdu 都表示一条短信消息。接着使用 SmsMessage 的 createFromPdu()方法将每一个 pdu 字节数组转换为 SmsMessage 对象，调用这个对象的 getOriginatingAddress()方法就可以获取到短信的发送方号码，调用 getMessageBody()方法就可以获取到短信的内容，然后将每一个 SmsMessage 对象中的短信内容拼接起来，就组成了一条完整的短信。最后将获取到的发送方号码和短信内容显示在 TextView 上。

完成了 MessageReceiver 之后，我们还需要对它进行注册才能让它接收到短信广播，代码如下所示：

```
public class MainActivity extends Activity {

    private TextView sender;

    private TextView content;

    private IntentFilter receiveFilter;

    private MessageReceiver messageReceiver;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        sender = (TextView) findViewById(R.id.sender);
        content = (TextView) findViewById(R.id.content);
        receiveFilter = new IntentFilter();
        receiveFilter.addAction("android.provider.Telephony.SMS_RECEIVED");
        messageReceiver = new MessageReceiver();
        registerReceiver(messageReceiver, receiveFilter);
    }

    @Override
```

```
    protected void onDestroy() {
        super.onDestroy();
        unregisterReceiver(messageReceiver);
    }
    ……
}
```

这些代码你应该都已经非常熟悉了，使用的就是动态注册广播的技术。在 onCreate()方法中对 MessageReceiver 进行注册，在 onDestroy()方法中再对它取消注册。

代码到这里就已经完成得差不多了，不过最后我们还需要给程序声明一个接收短信的权限才行，修改 AndroidManifest.xml 中的代码，如下所示：

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.smstest"
    android:versionCode="1"
    android:versionName="1.0" >
    <uses-permission android:name="android.permission.RECEIVE_SMS" />
    ……
</manifest>
```

现在可以来运行一下程序了，界面如图 8.5 所示。

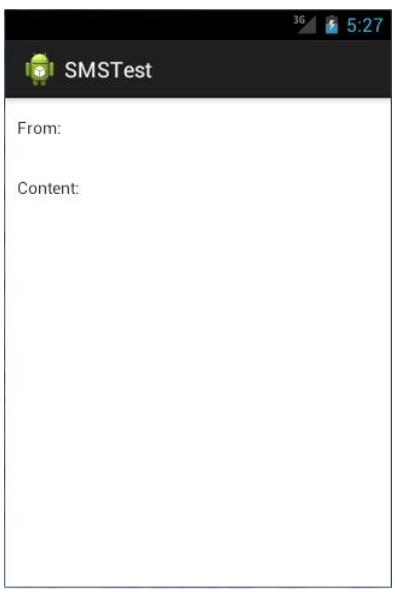

图　8.5

当有短信到来时，短信的发送方和内容就会显示在界面上。不过话说回来，我们使用的

是模拟器，模拟器上怎么可能会收得到短信呢？不用担心，DDMS 提供了非常充分的模拟环境，使得我们不需要支付真正的短信费用也可以模拟收发短信的场景。将 Eclipse 切换到 DDMS 视图下，然后点击 Emulator Control 切换卡，在这里就可以向模拟器发送短信了，如图 8.6 所示。

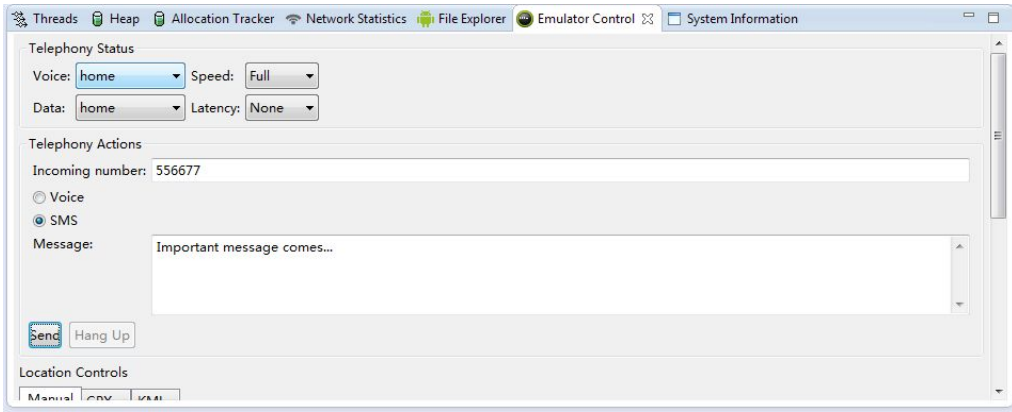

图 8.6

可以看到，我们指定发送方的号码是 556677，并填写了一段短信内容，然后点击 Send 按钮，这样短信就发送成功了。接着我们立马查看一下 SMSTest 这个程序，结果如图 8.7 所示。

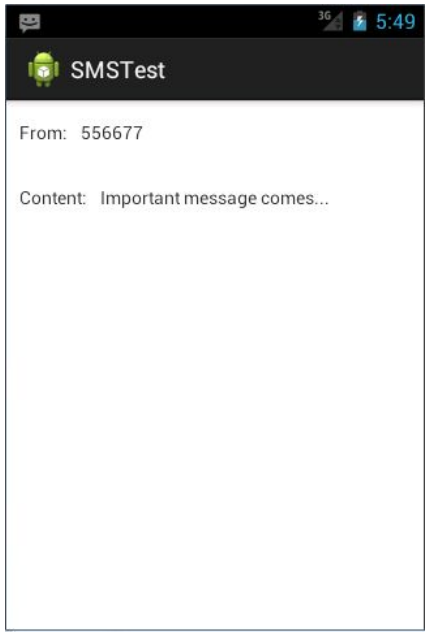

图　8.7

可以看到，短信的发送方号码和短信内容都显示到界面上了，说明接收短信的功能成功实现了。

8.2.2　拦截短信

仔细观察图 8.7，你会发现在系统状态栏出现了一个通知图标，这个通知图标是由 Android 自带的短信程序产生的。也就是说当短信到来时，不仅我们的程序会接收到这条短信，系统的短信程序同样也会收到。同样一条短信被重复接收两遍就会造成比较差的用户体验，那么有没有什么办法可以屏蔽系统短信程序的接收功能呢？

在前面 5.3.2 节学习有序广播的时候我们就已经知道，有序广播的传递是可以截断的，而系统发出的短信广播正是一条有序广播，因此这里我们的答案是肯定的。修改 MainActivity 中的代码，如下所示：

```
public class MainActivity extends Activity {
    ……
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        ……
        receiveFilter = new IntentFilter();
        receiveFilter.addAction("android.provider.Telephony.SMS_RECEIVED");
        receiveFilter.setPriority(100);
        messageReceiver = new MessageReceiver();
        registerReceiver(messageReceiver, receiveFilter);
    }
    ……
    class MessageReceiver extends BroadcastReceiver {

        @Override
        public void onReceive(Context context, Intent intent) {
            ……
            abortBroadcast();
        }

    }
}
```

可以看到，关键性的步骤只有两步。一是提高 MessageReceiver 的优先级，让它能够先于系统短信程序接收到短信广播。二是在 onReceive()方法中调用 abortBroadcast()方法，中止

掉广播的继续传递。

现在重新运行程序，再向模拟器发送一条短信，这时只有我们自己的程序才能收到这条短信了。按下 Back 键将程序关闭后，系统的短信程序又会重新拥有接收短信的功能。

注意这个功能一定要慎用，随意拦截短信有可能会造成重要数据的丢失，所以你在拦截之前一定要先想清楚这种功能是不是你想要的。

8.2.3 发送短信

下面我们继续对 SMSTest 项目进行扩展，给它加上发送短信的功能。那么还是先来编写一下布局文件吧，修改 activity_main.xml 中的代码，如下所示：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >
    ……
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="50dp" >
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_gravity="center_vertical"
            android:padding="10dp"
            android:text="To:" />

        <EditText
            android:id="@+id/to"
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:layout_gravity="center_vertical"
            android:layout_weight="1" />
    </LinearLayout>

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="50dp" >
        <EditText
            android:id="@+id/msg_input"
            android:layout_width="0dp"
```

```
            android:layout_height="wrap_content"
            android:layout_gravity="center_vertical"
            android:layout_weight="1" />

        <Button
            android:id="@+id/send"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_gravity="center_vertical"
            android:text="Send" />
    </LinearLayout>

</LinearLayout>
```

这里我们又新增了两个 LinearLayout，分别处于第三和第四行的位置。第三行中放置了一个 EditText，用于输入接收方的手机号码。第四行中放置了一个 EditText 和一个 Button，分别用于输入短信内容和发送短信。

然后修改 MainActivity 中的代码，在里面加入发送短信的处理逻辑，代码如下所示：

```
public class MainActivity extends Activity {
    ……
    private EditText to;

    private EditText msgInput;

    private Button send;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        ……
        to = (EditText) findViewById(R.id.to);
        msgInput = (EditText) findViewById(R.id.msg_input);
        send = (Button) findViewById(R.id.send);
        send.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
                SmsManager smsManager = SmsManager.getDefault();
                smsManager.sendTextMessage(to.getText().toString(), null,
```

```
                                        msgInput.getText().toString(), null, null);
                }
            });
        }
        ……
    }
```

可以看到，首先我们获取到了布局文件中新增控件的实例，然后在 Send 按钮的点击事件里面处理了发送短信的具体逻辑。当 Send 按钮被点击时，会先调用 SmsManager 的 getDefault()方法获取到 SmsManager 的实例，然后再调用它的 sendTextMessage()方法就可以去发送短信了。sendTextMessage()方法接收五个参数，其中第一个参数用于指定接收人的手机号码，第三个参数用于指定短信的内容，其他的几个参数我们暂时用不到，直接传入 null 就可以了。

接下来也许你已经猜到了，发送短信也是需要声明权限的，因此修改 AndroidManifest.xml 中的代码，如下所示：

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.smstest"
    android:versionCode="1"
    android:versionName="1.0" >
    <uses-permission android:name="android.permission.RECEIVE_SMS" />
    <uses-permission android:name="android.permission. SEND_SMS" />
    ……
</manifest>
```

现在重新运行程序之后，SMSTest 就拥有了发送短信的能力。不过点击 Send 按钮虽然可以将短信发送出去，但是我们并不知道到底发送成功了没有，这个时候就可以利用 sendTextMessage()方法的第四个参数来对短信的发送状态进行监控。修改 MainActivity 中的代码，如下所示：

```
public class MainActivity extends Activity {
    ……
    private IntentFilter sendFilter;

    private SendStatusReceiver sendStatusReceiver;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
```

```
        ……
        sendFilter = new IntentFilter();
        sendFilter.addAction("SENT_SMS_ACTION");
        sendStatusReceiver = new SendStatusReceiver();
        registerReceiver(sendStatusReceiver, sendFilter);
        send.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
                SmsManager smsManager = SmsManager.getDefault();
                Intent sentIntent = new Intent("SENT_SMS_ACTION");
                PendingIntent pi = PendingIntent.getBroadcast
(MainActivity.this, 0, sentIntent, 0);
                smsManager.sendTextMessage(to.getText().toString(), null,
msgInput.getText().toString(), pi, null);
            }
        });
    }

    @Override
    protected void onDestroy() {
        super.onDestroy();
        unregisterReceiver(messageReceiver);
        unregisterReceiver(sendStatusReceiver);
    }
    ……
    class SendStatusReceiver extends BroadcastReceiver {

        @Override
        public void onReceive(Context context, Intent intent) {
            if (getResultCode() == RESULT_OK) {
                // 短信发送成功
                Toast.makeText(context, "Send succeeded",
Toast.LENGTH_LONG).show();
            } else {
                // 短信发送失败
                Toast.makeText(context, "Send failed",
Toast.LENGTH_LONG).show();
            }
        }
```

```
        }
    }
```

可以看到，在 Send 按钮的点击事件里面我们调用了 PendingIntent 的 getBroadcast()方法获取到了一个 PendingIntent 对象，并将它作为第四个参数传递到 sendTextMessage()方法中。然后又注册了一个新的广播接收器 SendStatusReceiver，这个广播接收器就是专门用于监听短信发送状态的，当 getResultCode()的值等于 RESULT_OK 就会提示发送成功，否则提示发送失败。

现在重新运行一下程序，在文本输入框里输入接收方的手机号码以及短信内容，然后点击 Send 按钮，结果如图 8.8 所示。

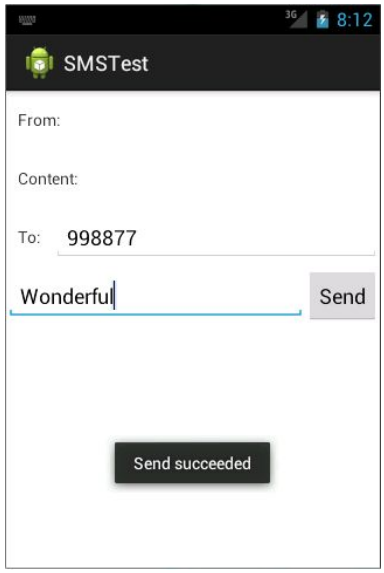

图 8.8

注意，这里虽然提示发送成功了，但实际上使用模拟器来发送短信对方是不可能收得到的，只有把这个项目运行在手机上，才能真正地实现发送短信的功能。

另外，根据国际标准，每条短信的长度不得超过 160 个字符，如果想要发送超出这个长度的短信，则需要将这条短信分割成多条短信来发送，使用 SmsManager 的 sendMultipartTextMessage()方法就可以实现上述功能。它的用法和 sendTextMessage()方法也基本类似，感兴趣的话你可以自己研究一下，这里就不再展开讲解了。

经验值：+15000　　目前经验值：76905
级别：资深鸟

赢得宝物：战胜短信鸟。拾取短信鸟掉落的宝物，神界 50 元 1 万条短信套餐一份、头戴式短信鸟专用信息输出器一个、三界飞行专用导航仪一个。短信鸟是一种小型鸟类，在神族分类学上属于通信科文字属短信种。所有的短信鸟之间都具有瞬间通讯能力（类似于人界的量子缠绕现象，但比那先进得多），而且短信鸟能看懂三界的所有语种的文字，这在神界低等神兽中是不多见的，至于短信鸟是怎么学会这些语种的，原因很简单，只是上古时代一位甚高阶天神在某一天关闭了短信鸟大脑中的“不能看懂所有语种文字”的开关。因为短信鸟不知道它应该看不懂这些语种的文字，因此它也就顺理成章地看懂了所有这些语种的文字。每只短信鸟都有一个唯一的名字，只要你把你要传递的信息写到纸上，然后写上远方另一只短信鸟的名字，让身边的短信鸟瞄上一眼，它就能把信息传递给名字所指定的远方的那只短信鸟，而远方的那只短信鸟可以用头戴式短信鸟专用信息输出器将信息显示到屏幕上供接受方阅读。神界的通讯工具有很多，但以生物形式存在的通讯工具唯有短信鸟，而且使用起来很有趣，很多家庭都拥有不只一只短信鸟。事实上，所有高级生物的大脑中都有一个“不能看懂所有语种文字”的开关，出生时是被开启的。但它可以被关闭。只是如何关闭它的方法早已失传，即使在神界这也是个未解之谜。尽管神们可以通过其他一些方法逐渐学会三界的所有语言，但无疑费时费力，而最好用的关闭开关法却已失传，不得不说是十分遗憾的事情。

8.3　调用摄像头和相册

前面两节所学习的知识当中，都涉及到了一些必须要在真正的 Android 手机上运行才看得到效果的功能。本节即将学习的知识也是，比如模拟器对摄像头的支持并不友好。你会发现，当涉及到多媒体这一领域的时候，模拟器多多少少会有些力不从心。因此，下面我们就先来学习一下，如何使用 Android 手机来运行程序。

8.3.1　将程序运行到手机上

不必我多说，首先你需要拥有一部 Android 手机。现在 Android 手机早就不是什么稀罕物，几乎已经是人手一部了，如果你还没有话，抓紧去购买吧。

想要将程序运行到手机上，我们需要先通过数据线把手机连接到电脑上。然后进入到设置→开发者选项界面，并在这个界面中勾选中 USB 调试选项，如图 8.9 所示。注意从 Android 4.2 版本开始，系统默认是把开发者选项隐藏掉的，你需要先进入到关于手机界面，然后对着最下面的版本号那一栏连击四次，就会让开发者选项显示出来。

图 8.9

然后如果你使用的是 Windows 操作系统，还需要在电脑上安装手机的驱动。一般借助 91 手机助手或豌豆荚等工具都可以快速地进行安装，安装完成后就可以看到手机已经连接到电脑上了，如图 8.10 所示。

图　8.10

现在进入到 Eclipse 的 DDMS 视图，你会发现当前是有两个设备在线的，一个是我们一直使用的模拟器，另外一个则是刚刚连接上的手机了，如图 8.11 所示。

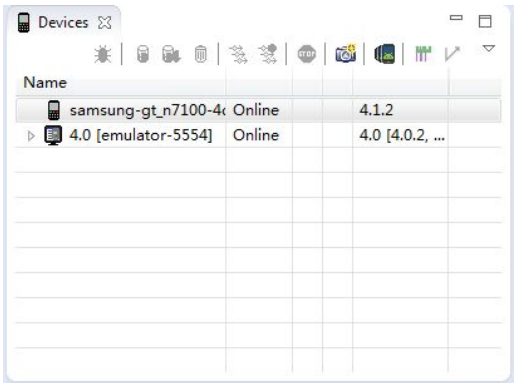

图　8.11

然后，对着 Eclipse 中的任何一个项目右击→Run As→Android Application，这时不会直接将程序运行到模拟器或者手机上，而是会弹出一个对话框让你进行选择，如图 8.12 所示。

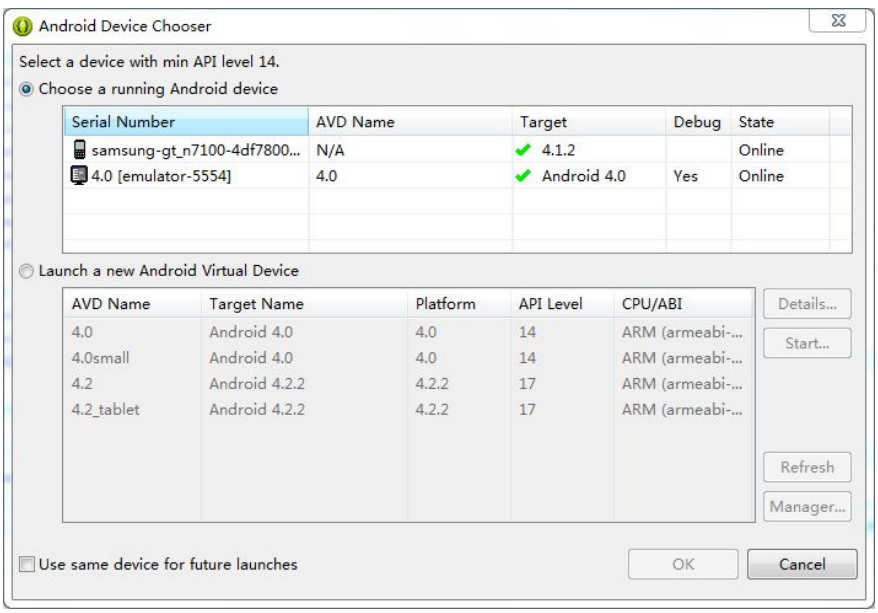

图　8.12

选择第一行的那个设备后点击 OK，就会将程序运行到手机上了。

8.3.2 调用摄像头拍照

很多应用程序都可能会使用到调用摄像头拍照的功能，比如说程序里需要上传一张图片作为用户的头像，这时打开摄像头拍张照是最简单快捷的。下面就让我们通过一个例子来学习一下，如何才能在应用程序里调用手机的摄像头进行拍照。

新建一个 ChoosePicTest 项目，然后修改 activity_main.xml 中的代码，如下所示：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >

    <Button
        android:id="@+id/take_photo"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Take Photo" />

    <ImageView
        android:id="@+id/picture"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center_horizontal" />

</LinearLayout>
```

可以看到，布局文件中只有两个控件，一个 Button 和一个 ImageView。Button 是用于打开摄像头进行拍照的，而 ImageView 则是用于将拍到的图片显示出来。

然后开始编写调用摄像头的具体逻辑，修改 MainActivity 中的代码，如下所示：

```
public class MainActivity extends Activity {

    public static final int TAKE_PHOTO = 1;

    public static final int CROP_PHOTO = 2;

    private Button takePhoto;

    private ImageView picture;
```

```
    private Uri imageUri;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        takePhoto = (Button) findViewById(R.id.take_photo);
        picture = (ImageView) findViewById(R.id.picture);
        takePhoto.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
                // 创建File对象，用于存储拍照后的图片
                File outputImage = new File(Environment.
getExternalStorageDirectory(), "tempImage.jpg");
                try {
                    if (outputImage.exists()) {
                        outputImage.delete();
                    }
                    outputImage.createNewFile();
                } catch (IOException e) {
                    e.printStackTrace();
                }
                imageUri = Uri.fromFile(outputImage);
                Intent intent = new Intent("android.media.action. IMAGE_CAPTURE");
                intent.putExtra(MediaStore.EXTRA_OUTPUT, imageUri);
                startActivityForResult(intent, TAKE_PHOTO); // 启动相机程序
            }
        });
    }

    @Override
    protected void onActivityResult(int requestCode, int resultCode, Intent data) {
        switch (requestCode) {
        case TAKE_PHOTO:
            if (resultCode == RESULT_OK) {
                Intent intent = new Intent("com.android.camera.action.CROP");
                intent.setDataAndType(imageUri, "image/*");
                intent.putExtra("scale", true);
```

```
                intent.putExtra(MediaStore.EXTRA_OUTPUT, imageUri);
                startActivityForResult(intent, CROP_PHOTO); // 启动裁剪程序
            }
            break;
        case CROP_PHOTO:
            if (resultCode == RESULT_OK) {
                try {
                    Bitmap bitmap = BitmapFactory.decodeStream
(getContentResolver()
                            .openInputStream(imageUri));
                    picture.setImageBitmap(bitmap); // 将裁剪后的照片显示出来
                } catch (FileNotFoundException e) {
                    e.printStackTrace();
                }
            }
            break;
        default:
            break;
        }
    }

}
```

上述代码稍微有点复杂，我们来仔细地分析一下。在 MainActivity 中要做的第一件事自然是分别获取到 Button 和 ImageView 的实例，并给 Button 注册上点击事件，然后在 Button 的点击事件里开始处理调用摄像头的逻辑，我们重点看下这部分代码。

首先这里创建了一个 File 对象，用于存储摄像头拍下的图片，这里我们把图片命名为 output_image.jpg，并将它存放在手机 SD 卡的根目录下，调用 Environment 的 getExternalStorageDirectory()方法获取到的就是手机 SD 卡的根目录。然后再调用 Uri 的 fromFile()方法将 File 对象转换成 Uri 对象，这个 Uri 对象标识着 output_image.jpg 这张图片的唯一地址。接着构建出一个 Intent 对象，并将这个 Intent 的 action 指定为 android.media.action.IMAGE_CAPTURE，再调用 Intent 的 putExtra()方法指定图片的输出地址，这里填入刚刚得到的 Uri 对象，最后调用 startActivityForResult()来启动活动。由于我们使用的是一个隐式 Intent，系统会找出能够响应这个 Intent 的活动去启动，这样照相机程序就会被打开，拍下的照片将会输出到 output_image.jpg 中。

注意刚才我们是使用 startActivityForResult()来启动活动的，因此拍完照后会有结果返回到 onActivityResult()方法中。如果发现拍照成功，则会再次构建出一个 Intent 对象，并把它的 action 指定为 com.android.camera.action.CROP。这个 Intent 是用于对拍出的照片进行裁剪

的，因为摄像头拍出的照片都比较大，而我们可能只希望截取其中的一小部分。然后给这个 Intent 设置上一些必要的属性，并再次调用 startActivityForResult()来启动裁剪程序。裁剪后的照片同样会输出到 output_image.jpg 中。

裁剪操作完成之后，程序又会回调到 onActivityResult()方法中，这个时候我们就可以调用 BitmapFactory 的 decodeStream()方法将 output_image.jpg 这张照片解析成 Bitmap 对象，然后把它设置到 ImageView 中显示出来。

由于这个项目涉及到了向 SD 卡中写数据的操作，因此我们还需要在 AndroidManifest.xml 中声明权限：

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.choosepictest"
    android:versionCode="1"
    android:versionName="1.0" >
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
    ……
</manifest>
```

这样代码就都编写完了，现在将程序运行到手机上，然后点击 Take Photo 按钮就可以进行拍照了，如图 8.13 所示。

图　8.13

拍照完成后点击确定则可以对照片进行裁剪，如图 8.14 所示。

图 8.14

点击完成，就会回到我们程序的界面。同时，裁剪后的照片当然也会显示出来了，如图8.15 所示。

图 8.15

8.3.3　从相册中选择照片

虽然调用摄像头拍照既方便又快捷，但并不是每一次我们都需要去当场拍一张照片的。因为每个人的手机相册里应该都会存有许许多多张照片，直接从相册里选取一张现有的照片会比打开相机拍一张照片更加常用。一个优秀的应用程序应该将这两种选择方式都提供给用户，由用户来决定使用哪一种。下面我们就来看一下，如何才能实现从相册中选择照片的功能。

还是在 ChoosePicTest 项目的基础上进行修改，首先编辑 activity_main.xml 文件，在布局中添加一个按钮用于从相册中选择照片，代码如下所示：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >

    <Button
        android:id="@+id/take_photo"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Take Photo" />

    <Button
        android:id="@+id/choose_from_album"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Choose From Album" />

    <ImageView
        android:id="@+id/picture"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center_horizontal" />

</LinearLayout>
```

然后修改 MainActivity 中的代码，加入从相册选择照片的逻辑，代码如下所示：

```
public class MainActivity extends Activity {
    ……
    private Button chooseFromAlbum;
```

```
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        ……
        chooseFromAlbum = (Button) findViewById(R.id.choose_from_album);
        chooseFromAlbum.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
                // 创建File对象，用于存储选择的照片
                File outputImage = new File(Environment.
getExternalStorageDirectory(), "output_image.jpg");
                try {
                    if (outputImage.exists()) {
                        outputImage.delete();
                    }
                    outputImage.createNewFile();
                } catch (IOException e) {
                    e.printStackTrace();
                }
                imageUri = Uri.fromFile(outputImage);
                Intent intent = new Intent("android.intent.action.
GET_CONTENT");
                intent.setType("image/*");
                intent.putExtra("crop", true);
                intent.putExtra("scale", true);
                intent.putExtra(MediaStore.EXTRA_OUTPUT, imageUri);
                startActivityForResult(intent, CROP_PHOTO);
            }
        });
    }
    ……
}
```

可以看到，在 Choose From Album 按钮的点击事件里我们同样创建了一个 File 对象，用于存储从相册中选择的图片。然后构建出一个 Intent 对象，并将它的 action 指定为 android.intent.action.GET_CONTENT。接着给这个 Intent 对象设置一些必要的参数，包括是否允许缩放和裁剪、图片的输出位置等。最后调用 startActivityForResult()方法，就可以打开相册程序选择照片了。

注意在调用 startActivityForResult()方法的时候，我们给第二个参数传入的值仍然是 CROP_PHOTO 常量，这样的好处就是从相册选择好照片之后，会直接进入到 CROP_PHOTO 的 case 下将图片显示出来，这样就可以复用之前写好的显示图片的逻辑，不用再编写一遍了。

现在将程序重新运行到手机上，然后点击一下 Choose From Album 按钮，就会打开相册程序，如图 8.16 所示。

图　8.16

然后随意选择一张照片就可以对它进行裁剪，如图 8.17 所示。

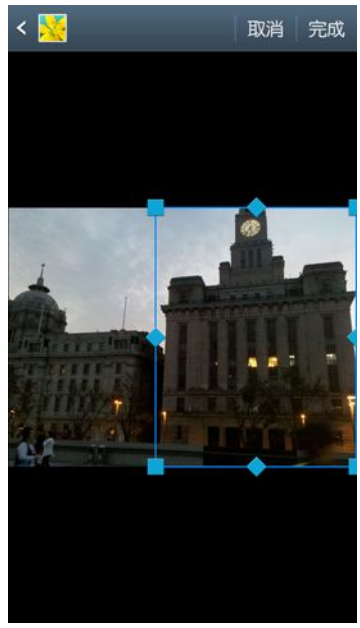

图 8.17

最后点击完成，回到我们程序的界面，就会看到裁剪后的图片已经显示出来了，如图8.18所示。

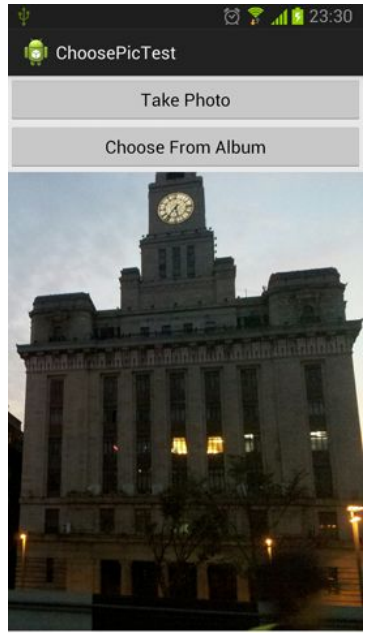

图　8.18

调用摄像头拍照以及从相册中选择照片是很多 Android 应用都会带有的功能，现在你已经将这两种技术都学会了，将来在工作中如果需要开发类似的功能，相信你一定能轻松完成吧。不过目前我们的实现还不算完美，因为某些照片即使经过裁剪后体积仍然很大，直接加载到内存中有可能会导致程序崩溃。更好的做法是根据项目的需求先对照片进行适当的压缩，然后再加载到内存中。至于如何对照片进行压缩，就要考验你查阅资料的能力了，这里就不再展开进行讲解。

8.4　播放多媒体文件

手机上最常见的休闲方式毫无疑问就是听音乐和看电影了，随着移动设备的普及，越来越多人都可以随时享受优美的音乐，以及观看精彩的电影。而 Android 在播放音频和视频方面也是做了相当不错的支持，它提供了一套较为完整的 API，使得开发者可以很轻松地编写出一个简易的音频或视频播放器，下面我们就来具体地学习一下。

8.4.1　播放音频

在 Android 中播放音频文件一般都是使用 MediaPlayer 类来实现的，它对多种格式的音频文件提供了非常全面的控制方法，从而使得播放音乐的工作变得十分简单。下表列出了 MediaPlayer 类中一些较为常用的控制方法。

方法名	功能描述
setDataSource()	设置要播放的音频文件的位置。
prepare()	在开始播放之前调用这个方法完成准备工作。
start()	开始或继续播放音频。
pause()	暂停播放音频。
reset()	将 MediaPlayer 对象重置到刚刚创建的状态。
seekTo()	从指定的位置开始播放音频。
stop()	停止播放音频。调用这个方法后的 MediaPlayer 对象无法再播放音频。
release()	释放掉与 MediaPlayer 对象相关的资源。
isPlaying()	判断当前 MediaPlayer 是否正在播放音频。
getDuration()	获取载入的音频文件的时长。

简单了解了上述方法后，我们再来梳理一下 MediaPlayer 的工作流程。首先需要创建出一个 MediaPlayer 对象，然后调用 setDataSource()方法来设置音频文件的路径，再调用 prepare()

方法使MediaPlayer进入到准备状态，接下来调用start()方法就可以开始播放音频，调用pause()方法就会暂停播放，调用reset()方法就会停止播放。

下面就让我们通过一个具体的例子来学习一下吧，新建一个PlayAudioTest项目，然后修改activity_main.xml中的代码，如下所示：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >

    <Button
        android:id="@+id/play"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="Play" />

    <Button
        android:id="@+id/pause"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="Pause" />

    <Button
        android:id="@+id/stop"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="Stop" />

</LinearLayout>
```

布局文件中横向放置了三个按钮，分别用于对音频文件进行播放、暂停和停止操作。然后修改MainActivity中的代码，如下所示：

```
public class MainActivity extends Activity implements OnClickListener {

    private Button play;

    private Button pause;
```

```
    private Button stop;

    private MediaPlayer mediaPlayer = new MediaPlayer();

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        play = (Button) findViewById(R.id.play);
        pause = (Button) findViewById(R.id.pause);
        stop = (Button) findViewById(R.id.stop);
        play.setOnClickListener(this);
        pause.setOnClickListener(this);
        stop.setOnClickListener(this);
        initMediaPlayer(); // 初始化MediaPlayer
    }

    private void initMediaPlayer() {
        try {
            File file = new File(Environment.getExternalStorageDirectory(),
"music.mp3");
            mediaPlayer.setDataSource(file.getPath()); // 指定音频文件的路径
            mediaPlayer.prepare(); // 让MediaPlayer进入到准备状态
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    @Override
    public void onClick(View v) {
        switch (v.getId()) {
        case R.id.play:
            if (!mediaPlayer.isPlaying()) {
                mediaPlayer.start(); // 开始播放
            }
            break;
        case R.id.pause:
            if (mediaPlayer.isPlaying()) {
                mediaPlayer.pause(); // 暂停播放
            }
```

```
            break;
        case R.id.stop:
            if (mediaPlayer.isPlaying()) {
                mediaPlayer.reset(); // 停止播放
                initMediaPlayer();
            }
            break;
        default:
            break;
        }
    }

    @Override
    protected void onDestroy() {
        super.onDestroy();
        if (mediaPlayer != null) {
            mediaPlayer.stop();
            mediaPlayer.release();
        }
    }

}
```

可以看到，在类初始化的时候我们就创建了一个 MediaPlayer 的实例，然后在 onCreate()方法中调用了 initMediaPlayer()方法为 MediaPlayer 对象进行初始化操作。在 initMediaPlayer()方法中，首先是通过创建一个 File 对象来指定音频文件的路径，从这里可以看出，我们需要事先在 SD 卡的根目录下放置一个名为 music.mp3 的音频文件。后面依次调用了 setDataSource()方法和 prepare()方法为 MediaPlayer 做好了播放前的准备。

接下来我们看一下各个按钮的点击事件中的代码。当点击 Play 按钮时会进行判断，如果当前 MediaPlayer 没有正在播放音频，则调用 start()方法开始播放。当点击 Pause 按钮时会判断，如果当前 MediaPlayer 正在播放音频，则调用 pause()方法暂停播放。当点击 Stop 按钮时会判断，如果当前 MediaPlayer 正在播放音频，则调用 reset()方法将 MediaPlayer 重置为刚刚创建的状态，然后重新调用一遍 initMediaPlayer()方法。

最后在 onDestroy()方法中，我们还需要分别调用 stop()和 release()方法，将与 MediaPlayer 相关的资源释放掉。

这样一个简易版的音乐播放器就完成了，现在将程序运行到手机上，界面如图 8.19 所示。

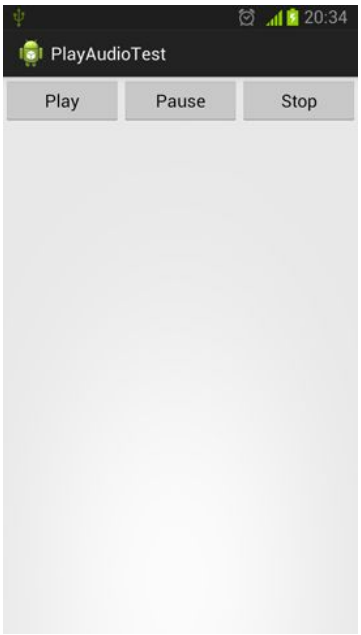

图　8.19

点击一下 Play 按钮就可以听到优美的音乐了，然后点击 Pause 按钮声音会停住，再次点击 Play 按钮会接着暂停之前的位置继续播放。这时如果点击一下 Stop 按钮声音也会停住，但是再次点击 Play 按钮时，音乐就会重头开始播放了。

8.4.2　播放视频

播放视频文件其实并不比播放音频文件复杂，主要是使用 VideoView 类来实现的。这个类将视频的显示和控制集于一身，使得我们仅仅借助它就可以完成一个简易的视频播放器。VideoView 的用法和 MediaPlayer 也比较类似，主要有以下常用方法：

方法名	功能描述
setVideoPath()	设置要播放的视频文件的位置。
start()	开始或继续播放视频。
pause()	暂停播放视频。
resume()	将视频重头开始播放。
seekTo()	从指定的位置开始播放视频。
isPlaying()	判断当前是否正在播放视频。
getDuration()	获取载入的视频文件的时长。

那么我们还是通过一个实际的例子来学习一下吧，新建 PlayVideoTest 项目，然后修改 activity_main.xml 中的代码，如下所示：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >

    <VideoView
        android:id="@+id/video_view"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" />

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

        <Button
            android:id="@+id/play"
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="Play" />

        <Button
            android:id="@+id/pause"
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="Pause" />

        <Button
            android:id="@+id/replay"
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="Replay" />
    </LinearLayout>

</LinearLayout>
```

在这个布局文件中，首先是放置了一个 VideoView，稍后的视频就将在这里显示。然后在 VideoView 的下面又放置了三个按钮，分别用于控制视频的播放、暂停和重新播放。

接下来修改 MainActivity 中的代码，如下所示：

```
public class MainActivity extends Activity implements OnClickListener {

    private VideoView videoView;

    private Button play;

    private Button pause;

    private Button replay;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        play = (Button) findViewById(R.id.play);
        pause = (Button) findViewById(R.id.pause);
        replay = (Button) findViewById(R.id.replay);
        videoView = (VideoView) findViewById(R.id.video_view);
        play.setOnClickListener(this);
        pause.setOnClickListener(this);
        replay.setOnClickListener(this);
        initVideoPath();
    }

    private void initVideoPath() {
        File file = new File(Environment.getExternalStorageDirectory(),
"movie.3gp");
        videoView.setVideoPath(file.getPath()); // 指定视频文件的路径
    }

    @Override
    public void onClick(View v) {
        switch (v.getId()) {
        case R.id.play:
            if (!videoView.isPlaying()) {
                videoView.start(); // 开始播放
```

```
            }
            break;
        case R.id.pause:
            if (videoView.isPlaying()) {
                videoView.pause(); // 暂时播放
            }
            break;
        case R.id.replay:
            if (videoView.isPlaying()) {
                videoView.resume(); // 重新播放
            }
            break;
        }
    }

    @Override
    protected void onDestroy() {
        super.onDestroy();
        if (videoView != null) {
            videoView.suspend();
        }
    }

}
```

这部分代码相信你理解起来会很轻松，因为它和前面播放音频的代码非常类似。首先在onCreate()方法中仍然是去获取一些控件的实例，然后调用了 initVideoPath()方法来设置视频文件的路径，这里我们需要事先在 SD 卡的根目录下放置一个名为 movie.3gp 的视频文件。

下面看一下各个按钮的点击事件中的代码。当点击 Play 按钮时会进行判断，如果当前并没有正在播放音频，则调用 start()方法开始播放。当点击 Pause 按钮时会判断，如果当前视频正在播放，则调用 pause()方法暂时播放。当点击 Replay 按钮时会判断，如果当前视频正在播放，则调用 resume()方法重头播放视频。

最后在 onDestroy()方法中，我们还需要调用一下 suspend()方法，将 VideoView 所占用的资源释放掉。

现在将程序运行到手机上，然后点击一下 Play 按钮，就可以看到视频已经开始播放了，如图 8.20 所示。

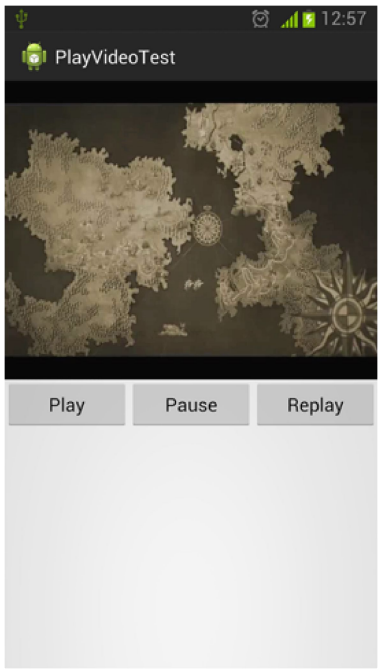

图　8.20

点击 Pause 按钮可以暂停视频的播放，点击 Replay 按钮可以重头播放视频。

这样的话，你就已经将 VideoView 的基本用法掌握得差不多了。不过，为什么它的用法和 MediaPlayer 这么相似呢？其实 VideoView 只是帮我们做了一个很好的封装而已，它的背后仍然是使用 MediaPlayer 来对视频文件进行控制的。另外需要注意，VideoView 并不是一个万能的视频播放工具类，它在视频格式的支持以及播放效率方面都存在着较大的不足。所以，如果想要仅仅使用 VideoView 就编写出一个功能非常强大的视频播放器是不太现实的。但是如果只是用于播放一些游戏的片头动画，或者某个应用的视频宣传，使用 VideoView 还是绰绰有余的。

好了，关于 Android 多媒体方面的知识你已经学得足够多了，下面就让我们一起来总结一下本章所学的内容吧。

8.5　小结与点评

本章我们主要对 Android 系统中的各种多媒体技术进行了学习，其中包括通知的使用技巧、调用摄像头拍照、从相册中选取照片，以及播放音频和视频文件。由于所涉及的多媒体技术在模拟器上很难看得到效果，因此本章中还特意讲解了在 Android 手机上调试程序的方

法。此外，我们还学习了如何使用 Android 提供的 API 来接收、发送和拦截短信，这使得我们甚至可以编写一个自己的短信程序来替换掉系统的短信程序。

又是充实饱满的一章啊！现在多媒体方面的知识已经学得足够多了，我希望你可以很好地将它们消化掉，尤其是与通知相关的内容，因为在下一章的学习当中我们还会用到它。在进行了一章多媒体相关知识的学习之后，你是否想起来 Android 四大组件中还剩一个没有学过呢，那么下面就让我们进入到 Android 服务的学习旅程之中。

经验值：+15000　　目前经验值：91905

级别：资深鸟

赢得宝物：战胜多媒体神。多媒体神身高 3 米，是一位打扮得非常奇特的潮神，事实上除了亲戚朋友，很少有人知道多媒体神本人到底长啥样，因为每次他出现在众人面前时，总是全副武装的，脑袋上顶着 24 个探照灯，腰部以上安装着 76 个大大小小的显示器，腰部以下则密不透风的挂满了 246 个大大小小的音箱，而且所有的显示器和音像都在播放完全不同的视频、电影、广告片、流行歌曲，以及交响乐。他声称，作为多媒体神，他一直在努力诠释多媒体的真谛——多！尽管大多数人觉得他有些扰民，但多媒体神脾气很好，他建立了多个慈善组织，帮助人界和魔界的有困难的生灵。神界生活安定富足，故没有慈善组织。