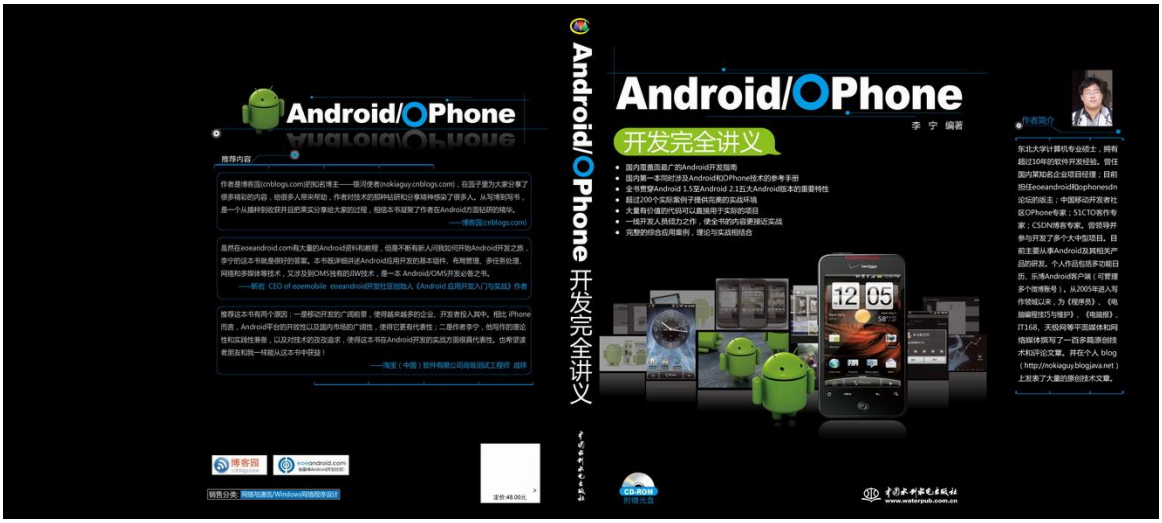
《Android/OPhone 开发完全讲义》样章

李宁　编著

http://nokiaguy.blogjava.net



内容简介

本书近 500 页，共 25 章，分为 5 篇，超过 200 个完整的例子、超过 2 万行代码。

第 1 篇：介绍了 Android 的相关内容，其中包括 Android 的基本概念、搭建 Android 开发环境、Android SDK 中常用命令行工具的使用方法、在 PC 上安装 Android，并测试程序、Android 的学习资源等，还会给出一个简单的 Android 程序，使读者可以了解开发 Android 程序的基本过程。

第 2 篇：主要讲解了 Android 的用户接口，包括 View、定制组件、对话框、Toast 和 Notification、各种菜单、布局；Android 中中的组件（android.widget 包中的组件）；移动存储解决方案，包括 SharedPreferences、PreferenceActivity、文件存储、XML 存储、Sqlite 数据库及其应用、ContentProvider 等；应用程序之间的通讯，包括跨进程调用 Activity、接收和发送广播；Android 服务详细解，包括 Service 的生命周期，系统服务、时间服务和 AIDL 服务；网络，包括从网络上获得数据装载 Gallery、ListView 等组件。以及从 google 上获得图像，WebView 组件、访问 Http 资源以及 Webservice 等；多媒体技术，包括图形的基本操作、旋转图像、透明度、扭曲和拉伸图像、路径，音频和视频等技术。

第 3 篇：2D 和 OpenGL ES 动画，国际化，Android 中的 14 种资源，访问 Android 手机的硬件，包括通过 USB 驱动在手机上直接调试程序，录音，控制手机的摄像头、传感器（电子罗盘，计步器），GPS 和地图定位，wifi 等。App Widget、在桌面上添加快捷方式，实时文件夹（LiveFolder），NDK，脚本语言（Python、perl 等），手势输入、TTS、蓝牙等。

第 4 篇：介绍了 OPhone 的基础知识，以及 OPhone 与 Android 的差异，OPhone 的 API 扩展，并详细介绍了 JIL Widget 在多媒体，获得系统信息，控制硬件等方面的内容。

第 5 篇：给出了两个综合的例子：万年历，知道当前位置的 GTalk 机器人。

14

访问 Android 手机的硬件

现在手机已不仅仅是打电话的工具了，从早期的手机引入摄像头开始，各种类型的硬件逐渐在手机中出现。例如传感器、GPS、WIFI、蓝牙等设备在手机中屡见不鲜。而作为开发人员的我们，不仅要理解和使用这些硬件，还要学会用程序随心所欲地控制它们。如果你也和作者的想法一致，那么本章将会给你带来意想不到的惊喜。

本章内容

- 在手机上测试、调试程序
- 录音
- 调用系统提供的拍照功能
- 实现自己的拍照功能
- 方向传感器和加速传感器
- 电子罗盘
- 计步器
- Google 地图
- GPS 定位
- WIFI

14.1 在手机上测试硬件

虽然 Android 模拟器可以测试大多数应用程序，但却无法测试那些和硬件相关的程序，例如录音、拍照（虽然拍照的部分功能能在模拟器上运行，但并不是真正使用摄像头进行拍摄）、重力感应、GPS、WIFI

等。如果这些功能无法在模拟器上测试，将给开发工作带来非常大的困难。在发布 Android SDK 的同时发布了一个 Android USB 驱动。将手机和电脑通过数据线相连后，并在计算机上安装这个 Android USB 驱动，就可以将手机变成一个测试程序的模拟器，也就是说，在 Eclipse 中运行程序后，会直接在手机上运行，而不是在计算机的模拟器中运行，这样就可以得到真实的运行效果。

14.1.1　安装 Android USB 驱动

Android USB 驱动只有 Windows 才需要安装。如果读者在 MAC OS X 或 Linux 上测试 Android 应用程序，可以访问如下地址查看 Android USB 驱动的配置过程。本节只介绍 Windows 下的 Android USB 驱动的安装过程。

http://developer.android.com/intl/zh-CN/guide/developing/device.html#setting-up

在作者写作本书时，Android USB 驱动的最新版本是 Revision 3。该版本支持 Windows XP 和 Windows Vista。如果读者下载并安装了最新版的 Android SDK（下载地址如下），会自动将 USB 驱动安装在电脑上。

http://developer.android.com/intl/zh-CN/sdk/index.html

在安装完 Android SDK 后，读者会在<Android SDK 安装目录>中看到一个 usb_driver 目录，该目录就是 USB 驱动的安装目录。目录结构如图 14.1 所示。

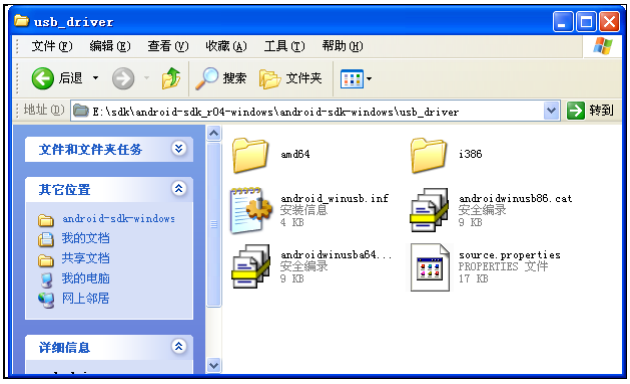


图 14.1　USB 驱动安装目录的结构

在安装完 Android USB 驱动后，还要根据手机的不同型号安装随机带的驱动程序。例如，作者使用的手机型号是 HTC Hero（G3），可以在 HTC 的官方网站上下载相应的驱动程序（HTC Sync）。成功安装驱动程序后，读者会在设备管理的右侧中看到 Android USB Drivers 项，如图 14.2 所示。

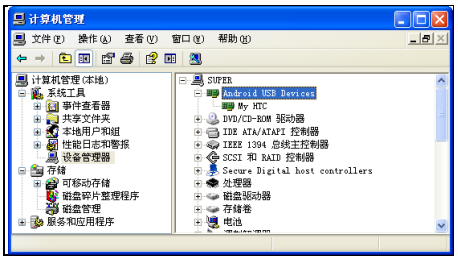


图 14.2　Android USB Drivers

在安装 Android SDK 时如果安装失败，可以切换到 Settings 列表项，并选中右侧界面下方的第 1 个复选项，如图 14.3 所示。

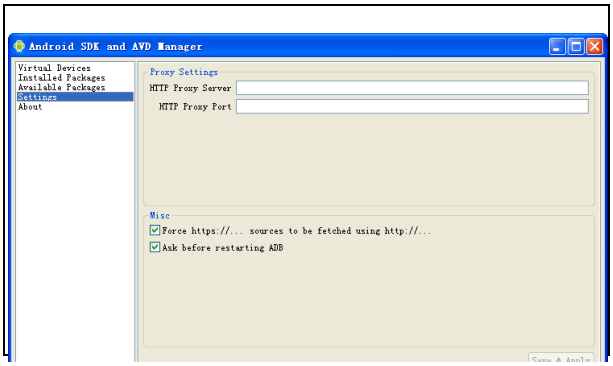


图 14.3　改成 http 下载

在安装完 Android USB 驱动后，需要进入手机的【设置】>【应用程序】>【开发】设备界面，选中【USB 调试】复选框，如图 14.4 所示。

14.1.2　在手机上测试程序

按 14.1.1 节介绍的成功安装 Android USB 驱动后，现在可以使用数据线连接手机和电脑了。如果手机中有 SD 卡，会在【我的电脑】中多一个【可移动磁盘】选项，不过这个可移动磁盘是否可访问，我们都不用管它。

现在可以用 Eclipse 运行程序了。如果这时未启动模拟器，程序会直接通过数据线传到手机上运行。如果已启动了模拟器，在运行程序之前，Eclipse 会弹出对话框要求开发人员选择在模拟器或在手机上运行程序，如图 14.5 所示。

图 14.4　选中【USB 调试】复选框（HTC Hero）

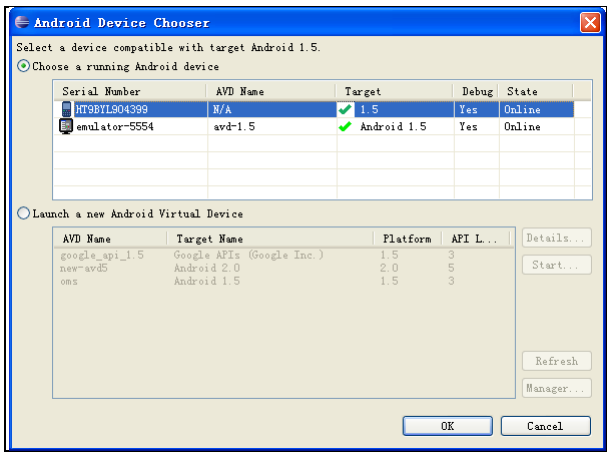


图 14.5　选择运行 Android 应用程序的设备

在如图 14.5 所示的界面中第 1 个设备是手机，第 2 个设备是模拟器。可以选择一个前面章节曾编写过的例子来做一个测试。例如选择第 11 章的实例 73（旋转的星系）进行测试，在手机上的运行效果如图 14.6 所示。

也许很多读者会注意到图 14.6 也是一个类似于模拟器的屏幕截图。只是这个截图来自真实的手机屏幕。从这一点可以看出，将手机变成可测试程序的模拟器后，可以和 PC 上的模拟器一样对手机屏幕进行截屏。读者可以在 DDMS 透视图中通过【Devices】视图的【Screen Capture】按钮测试截取手机屏幕的功能。但要注意，一定要选择【Devices】视图中的手机设备。

使用手机测试 Android 应用程序需要注意如下 2 点：

- 在【Run Configurations】对话框中不要选择运行目标（Target）的任何一个 AVD 设备，而且要将运行模式设为 Automatic，这样在运行程序时才会弹出选择运行设备的对话框。如果选中任何一个 AVD 设备（在 AVD 设备列表中并未列出手机设备），Eclipse 就会直接在选中的设备中运行程

序，而不会在手机上运行程序了。

- 如果在手机上已经安装了未使用 USB 驱动方式安装的程序（直接采用在手机上运行 apk 文件的方式安装应用程序），而且 package 与要运行的程序 package 相同，应先在手机上卸载该应用程序，然后通过 USB 驱动在手机上安装并测试程序。

14.1.3　在手机上调试程序

在手机上也可以调试应用程序。首先应在代码的相应位置设置断点，然后以 Debug 模式启动应用程序。如果代码执行到断点处，会在手机屏幕上弹出一个对话框，如图 14.7 所示。读者并不需要关闭这个对话框，几秒后该对话框会自动关闭。这时就可以像在模拟器中运行程序一样按步跟踪代码了。

图 14.6　旋转的星系（HTC Hero）

图 14.7　在手机上调试应用程序（HTC Hero）

14.2　录音

本节的例子代码所在的工程目录是 src\ch14\ch14_recorder

在模拟器中无法利用电脑的声卡录音，因此这个功能必须在真机上测试。录音功能需要使用 android.media.MediaRecorder 来完成。使用 MediaRecorder 录音需要通过如下 6 步完成：

（1）设置音频来源（一般为麦克风）。

（2）设置音频输出格式。

（3）设置音频编码方式。

（4）设置输出音频的文件名。

（5）调用 MediaRecorder 类的 prepare 方法。

（6）调用 MediaRecorder 类的 start 方法开始录音。

实现录音功能的完整代码如下：

```
MediaRecorder mediaRecorder = new MediaRecorder();
//  第 1 步：设置音频来源（MIC 表示麦克风）
mediaRecorder.setAudioSource(MediaRecorder.AudioSource.MIC);
//  第 2 步：设置音频输出格式（默认的输出格式）
mediaRecorder.setOutputFormat(MediaRecorder.OutputFormat.DEFAULT);
//  第 3 步：设置音频编码方式（默认的编码方式）
mediaRecorder.setAudioEncoder(MediaRecorder.AudioEncoder.DEFAULT);
//  创建一个临时的音频输出文件
audioFile = File.createTempFile("record_", ".amr");
//  第 4 步：指定音频输出文件
mediaRecorder.setOutputFile(audioFile.getAbsolutePath());
//  第 5 步：调用 prepare 方法
mediaRecorder.prepare();
```

```
//  第 6 步：调用 start 方法开始录音
mediaRecorder.start();
```

上面的代码指定了一个临时的音频输出文件，这就意味着每次将生成不同的音频文件。文件名的格式是 record_N.amr，其中 N 是整数。在录完音后，读者在 SD 卡的根目录会看到很多这样的文件（由录音的次数多少决定 amr 文件的多少）。

停止录单可以使用 MediaRecorder 类的 stop 方法，代码如下：

```
mediaRecorder.stop();
```

在生成 amr 文件后，可以使用 MediaPlayer 来播放 amr 文件。MediaPlayer 类的使用方法请读者参阅 10.2.1 节的内容。

14.3　控制手机摄像头（拍照）

现在几乎所有的手机都配有摄像头。而且随着摄像头的分辨率不断提高（有 200 万像素，有的可达到 500 万，甚至是 1000 万像素），用手机照相已成为很多用户最喜欢的方式。这主要是因为手机是唯一可以随时携带的电子设备。而数码相机虽然在拍照功能上比同档次的拍照手机更强大，但毕竟数码相机不能总是带在身边。除此之外，由于目前带摄像头的手机大多都是智能手机，除了拥有简单的拍照功能外，还可以通过程序对拍照的过程和拍摄后的图像进行处理。这样的功能远比数码相机要强大得多。而随时随地写微博也逐渐成为围脖（微博的斜音）们的时尚首选，而配上实时的照片将会为自己的微博吸引更多的粉丝，那么实现这些功能非手机莫属。读到这里，也许很多读者迫不急待地想在自己的应用程序中添加拍照功能，这些读者一定会对本节非常地感兴趣。

14.3.1　调用系统的拍照功能

本节的例子代码所在的工程目录是 src\ch14\ch14_systemcamera

读者可以先试试自己手机上的拍照功能。可能由于手机型号不同，拍照的方式和过程也可能不一样。在 HTC Hero 手机上进行拍照会由系统自动对焦，在对焦的过程中，屏幕上会出现一个白色的对焦符号（类似于中括号）。如果对焦成功，这个对焦符号就会变成绿色，如图 14.8 所示。

当对焦成功后，按手机下方的【呼吸灯】按钮进行拍照。在拍照后手机屏幕下方会出现两个按钮：【完成】和【拍照】按钮。如果对照片满意，单击【完成】按钮结束拍照。如果对照片不满意，单击【拍照】按钮继续拍照，上一次拍的照片将丢失。由于这两个按钮无法通过 DDMS 透视图截获，因此，只能截获所拍的照片，如图 14.9 所示。当完成拍照后，可以对照片做进一步处理，例如本节的例子将照片显示在 ImageView 中，如图 14.10 所示。

图 14.8　对焦成功（HTC Hero）

图 14.9　拍照成功（HTC Hero）

图 14.10　在 ImageVie 中显示照片（HTC Hero）

从上面的拍照过程可以猜到，用于显示拍照过程影像的界面实际上也是一个 Activity。因此要调用系

统的拍照功能，就要用到 7.1.2 节介绍的调用其他应用程序的 Activity 的方式。与拍照功能对应的 Action 是 android.provider.MediaStore.ACTION_IMAGE_CAPTURE。用于拍照的 Activity 需要返回照片图像数据，因此，需要使用 startActivityForResult 方法启动这个 Activity，代码如下：

```
Intent intent = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
startActivityForResult(intent, 1);
```

截获 Activity 返回的图像数据的事件方法是 onActivityResult，代码如下：

```
protected void onActivityResult(int requestCode, int resultCode, Intent data)
{
    if (requestCode == 1)
    {
        if (resultCode == Activity.RESULT_OK)
        {
            //  拍照 Activity 保存图像数据的 key 是 data，返回的数据类型是 Bitmap 对象
            Bitmap cameraBitmap = (Bitmap) data.getExtras().get("data");
            //  在 ImageView 组件中显示拍摄的照片
            imageView.setImageBitmap(cameraBitmap);
        }
    }
    super.onActivityResult(requestCode, resultCode, data);
}
```

在默认情况下，系统的拍照 Activity 将照片保存在 SD 卡的 DCIM\100MEDIA 目录中（不同型号的手机可能保存的目录不同）。在拍照的过程中按手机下方的【menu】按钮会在屏幕的下方显示几个选项菜单。单击【分辨率】菜单项，会弹出一个只有一个分辨率选项的对话框（在 HTC Hero 手机上的分别率是 624×416，如图 14.11 所示。这个分辨率可能随着手机型号的不同而不同，但分辨率都很小）。这就意味着所拍摄的照片分辨率不能大于 624*416。如果将照片保存成大于这个分辨率，照片就会失真。而手机自带的拍照程序可以根据手机摄像头的最大分辨率设置多个照片分辨率，如图 14.12 所示。

图 14.11　拍照 Activity 时可设置的照片分辨率（HTC Hero）

图 14.12　拍照程序可设置的照片分辨率（HTC Hero）

根据官方文档的解释，在调用拍照 Activity 时通过 MediaStore.EXTRA_OUTPUT 指定照片保存的路径，可以允许拍摄分辨率更大的照片。原文如下：

The caller may pass an extra EXTRA_OUTPUT to control where this image will be written. If the EXTRA_OUTPUT is not present, then a small sized image is returned as a Bitmap object in the extra field. This is useful for applications that only need a small image. If the EXTRA_OUTPUT is present, then the full-sized image will be written to the Uri value of EXTRA_OUTPUT.

按着官方的解释，可以使用如下代码调用拍照 Activity：

```
Intent intent = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
intent.putExtra(MediaStore.EXTRA_OUTPUT, Uri.fromFile(new File("/sdcard/test.jpg")));
startActivityForResult(intent, 1);
```

但经作者测试，在调用拍照 Activity 时设置 EXTRA_OUTPUT 并不起任何作用（仅对 Android SDK 1.5）。

这也许是 Android SDK 1.5 的一个 bug，或官方文档描述有误。如果读者非要拍摄更大分辨率的照片，可以实现自己的拍照 Activity 来完成这个功能，这部分内容将在 14.3.2 节介绍。

虽然使用系统的拍照 Activity 无法拍摄更大分辨率的照片，但可以同时生成分辨率更小的照片。通过 insertImage 方法可以同时在/sdcard/DCIM/.thumbnails 和/sdcard/DCIM/Camera 目录中分别生成分辨率为 50×50 和 208×312 的图像（其他型号的手机也有可能是其他的分辨率）。调用 insertImage 方法的代码如下：

```
MediaStore.Images.Media.insertImage(getContentResolver(), cameraBitmap, null, null);
```

其中 cameraBitmap 是拍照 Activity 返回的 Bitmap 对象。

不仅可以调用系统的拍照 Activity，而且可以调用系统的摄像 Activity。摄像 Activity 对应的 Action 是 MediaStore.ACTION_VIDEO_CAPTURE，调用方法与调用系统的拍照 Activity 相同。

14.3.2　实现自己的拍照 Activity

本节的例子代码所在的工程目录是 src\ch14\ch14_camera

拍照的核心类是 android.hardware.Camera，通过 Camera 类的静态方法 open 可以获得 Camera 对象，并通过 Camera 类的 startPreview 方法开始拍照，最后通过 Camera 类的 takePicture 方法结束拍照，并在相应的事件中处理照片数据。

上述的过程只是拍照过程的简化。在拍照之前，还需要做如下的准备工作。

- 指定用于显示拍照过程影像的容器，通常是 SurfaceHolder 对象。由于影像需要在 SurfaceView 对象中显示，因此可以使用 SurfaceView 类的 getHolder 方法获得 SurfaceHolder 对象。
- 在拍照过程中涉及到一些状态的变化。这些状态包括开始拍照（对应 surfaceCreated 事件方法）；拍照状态变化（例如图像格式或方向，对应 surfaceChanged 事件方法）；结束拍照（对应 surfaceDestroyed 事件方法）。这 3 个事件方法都是在 SurfaceHolder.Callback 接口中定义的，因此，需要使用 SurfaceHolder 接口的 addCallback 方法指定 SurfaceHolder.Callback 对象，以便捕捉这 3 个事件。
- 拍完照后需要处理照片数据。处理这些数据的工作需要在 PictureCallback 接口的 onPictureTaken 方法中完成。当调用 Camera 类的 takePicture 方法后，onPictureTaken 事件方法被调用。
- 如果需要自动对焦，需要调用 Camera 类的 autoFocus 方法。该方法需要一个 AutoFocusCallback 类型的参数值。AutoFocusCallback 是一个接口，在该接口中定义了一个 onAutoFocus 方法，当摄像头正在对焦或对焦成功都会调用该方法。

为了使拍照功能更容易使用，本节的例子将拍照功能封装在了 Preview 类中，代码如下：

```
class Preview extends SurfaceView implements SurfaceHolder.Callback
{
    private SurfaceHolder holder;
    private Camera camera;
    //  创建一个 PictureCallback 对象，并实现其中的 onPictureTaken 方法
    private PictureCallback pictureCallback = new PictureCallback()
    {
        //  该方法用于处理拍摄后的照片数据
        @Override
        public void onPictureTaken(byte[] data, Camera camera)
        {
            //  data 参数值就是照片数据，将这些数据以 key-value 形式保存，以便其他调用该 Activity 的程序可
            //  以获得照片数据
            getIntent().putExtra("bytes", data);
            setResult(20, getIntent());
            //  停止照片拍摄
            camera.stopPreview();
```

```
            camera = null;
            //  关闭当前的 Activity
            finish();
        }
    };
    //  Preview 类的构造方法
    public Preview(Context context)
    {
        super(context);
        //  获得 SurfaceHolder 对象
        holder = getHolder();
        //  指定用于捕捉拍照事件的 SurfaceHolder.Callback 对象
        holder.addCallback(this);
        //  设置 SurfaceHolder 对象的类型
        holder.setType(SurfaceHolder.SURFACE_TYPE_PUSH_BUFFERS);
    }
    //  开始拍照时调用该方法
    public void surfaceCreated(SurfaceHolder holder)
    {
        //  获得 Camera 对象
        camera = Camera.open();
        try
        {
            //  设置用于显示拍照影像的 SurfaceHolder 对象
            camera.setPreviewDisplay(holder);
        }
        catch (IOException exception)
        {
            //  释放手机摄像头
            camera.release();
            camera = null;
        }
    }
    //  停止拍照时调用该方法
    public void surfaceDestroyed(SurfaceHolder holder)
    {
        //  释放手机摄像头
        camera.release();
    }
    //  拍照状态变化时调用该方法
    public void surfaceChanged(final SurfaceHolder holder, int format, int w, int h)
    {
        try
        {
            Camera.Parameters parameters = camera.getParameters();
            //  设置照片格式
            parameters.setPictureFormat(PixelFormat.JPEG);
            // 根据屏幕方向设置预览尺寸
            if (getWindowManager().getDefaultDisplay().getOrientation() == 0)
                parameters.setPreviewSize(h, w);
            else
                parameters.setPreviewSize(w, h);
            //  设置拍摄照片的实际分辨率，本例中的分辨率是 1024×768
            parameters.setPictureSize(1024, 768);
            // 设置保存的图像大小
            camera.setParameters(parameters);
            //  开始拍照
            camera.startPreview();
            //  准备用于表示对焦状态的图像（类似图 14.8 所示的对焦符号）
            ivFocus.setImageResource(R.drawable.focus1);
            LayoutParams layoutParams = new LayoutParams(
                    LayoutParams.FILL_PARENT, LayoutParams.FILL_PARENT);
            ivFocus.setScaleType(ScaleType.CENTER);
```

```
                addContentView(ivFocus, layoutParams);
                ivFocus.setVisibility(VISIBLE);
                // 自动对焦
                camera.autoFocus(new AutoFocusCallback()
                {
                    @Override
                    public void onAutoFocus(boolean success, Camera camera)
                    {
                        if (success)
                        {
                            //  success 为 true 表示对焦成功，改变对焦状态图像（一个绿色的 png 图像）
                            ivFocus.setImageResource(R.drawable.focus2);
                        }
                    }
                });
            }
            catch (Exception e)
            {
            }
        }
        //  停止拍照，并将拍摄的照片传入 PictureCallback 接口的 onPictureTaken 方法
        public void takePicture()
        {
            if (camera != null)
            {
                camera.takePicture(null, null, pictureCallback);
            }
        }
}
```

在编写 Preview 类时应注意如下 7 点：

- 由于 Preview 是 CameraPreview 的内嵌类（CameraPreview 就是自定义的拍照 Activity）。因此，在 Preview 类中通过 putExtra 方法保存的数据会在调用 CameraPreview 的类中通过 onActivityResult 事件方法获得。
- Camera 类的 takePicture 方法有 3 个参数，都是回调对象，但比较常用的是最后一个参数。当拍完照后会调用该参数指定对象中的 onPictureTaken 方法，一般可以在该方法中对照片数据做进一步处理。例如，在本例中使用 putExtra 方法以 key-value 对保存了照片数据。
- 当手机摄像头的状态变化时，例如手机由纵向变成横向，或分辨率发生变化后，很多参数需要重新设置，这时系统就会调用 SurfaceHolder.Callback 接口的 surfaceChanged 方法。因此，可以在该方法中对摄像头的参数进行设置，包括调用 startPreview 方法进行拍照。
- 根据手机的拍摄方向（纵向或横向），需要设置预览尺寸。surfaceChanged 方法的最后两个参数表示摄像头预览时的实际尺寸。在使用 Camera.Parameters 类的 setPreviewSize 方法设置预览尺寸时，如果是纵向拍摄，setPreviewSize 方法的第 1 个参数值是 h，第 2 个参数值是 w，如果是横向拍摄，第 1 个参数值是 w，第 2 个参数值是 h。在设置时千万不要弄错了，否则当手机改变拍摄方向时无法正常拍照。读者可以改变 Preview 类中的预览尺寸，看看会产生什么效果。
- 如果想设置照片的实际分辨率，需要使用 Camera.Parameters 类的 setPictureSize 方法进行设置。
- 本例中通过在 CameraActivity 中添加 ImageView 的方式在预览界面显示了一个表示对焦状态的图像。这个图像文件有两个：focus1.png 和 focus2.png。其中 focus1.png 是白色的透明图像，表示正在对焦。focus2.png 是绿色的透明图像，表示对焦成功。在开始拍照后，先显示 focus1.png，当对焦成功后，系统会调用 AutoFocusCallback 接口的 onAutoFocus 方法。在该方法中将 ImageView 中显示的图像变成 focus2.png，表示对焦成功，这时就可以结束拍照了。
- 在拍完照后需要调用 Camera 类的 release 方法释放手机摄像头，否则除非重启手机，否则其他的应用程序无法再使用摄像头进行拍照。

本例通过触摸拍照预览界面结束拍照。因此，需要使用 Activity 的 onTouchEvent 事件方法来处理屏幕触摸事件，代码如下：

```
public boolean onTouchEvent(MotionEvent event)
{
    if (event.getAction() == MotionEvent.ACTION_DOWN)
        //  结束拍照
        preview.takePicture();
    return super.onTouchEvent(event);
}
```

其中 preview 是 Preview 类的对象实例，在 CameraPreview 类的 onCreate 方法中创建了该对象。

在编写完 CameraPreview 类后，可以在其他的类中使用如下代码启动 CameraPreview，启动 CameraPreview 后会自动进行拍照：

```
Intent intent = new Intent(this, CameraPreview.class);
startActivityForResult(intent, 1);
```

在关闭 CameraPreview 后（可能是拍照成功，也可能是取消拍照），可以通过 onActivityResult 方法来获得成功拍照后的照片数据，代码如下：

```
protected void onActivityResult(int requestCode, int resultCode, Intent data)
{
    if (requestCode == 1)
    {
        //  拍照成功后，响应码是 20
        if (resultCode == 20)
        {
            Bitmap cameraBitmap;
            //  获得照片数据（byte 数组形式）
            byte[] bytes = data.getExtras().getByteArray("bytes");
            //  将 byte 数组转换成 Bitmap 对象
            cameraBitmap = BitmapFactory.decodeByteArray(bytes, 0,bytes.length);
            //  根据拍摄的方向旋转图像（纵向拍摄时需要将图像旋转 90 度）
            if (getWindowManager().getDefaultDisplay().getOrientation() == 0)
            {
                Matrix matrix = new Matrix();
                matrix.setRotate(90);
                cameraBitmap = Bitmap.createBitmap(cameraBitmap, 0, 0,
                        cameraBitmap.getWidth(), cameraBitmap.getHeight(),matrix, true);
            }
            //  将照片保存在 SD 卡的根目录（文件名是 camera.jpg）
            File myCaptureFile = new File("/sdcard/camera.jpg");
            try
            {
                BufferedOutputStream bos = new BufferedOutputStream(
                        new FileOutputStream(myCaptureFile));
                cameraBitmap.compress(Bitmap.CompressFormat.JPEG, 100, bos);
                bos.flush();
                bos.close();
                imageView.setImageBitmap(cameraBitmap);
            }
            catch (Exception e)
            {
            }
        }
    }
    super.onActivityResult(requestCode, resultCode, data);
}
```

在编写上面代码时应注意如下两点：

- 由于纵向拍摄时生成的照片是横向的，因此需要在处理照片时将其顺时针旋转 90 度。在 14.3.1 节介绍的系统拍照 Activity 已经将照片处理完了，因此，不需要对照片进行旋转。
- 由于直接使用 Camera 类进行拍照时，系统不会自动保存照片，因此，就需要在处理照片时自行

确定照片的存储位置，并保存照片。这种方法的优点是灵活，缺点是需要写更多的代码。至于是选择系统提供的拍照功能，还是选择自己实现拍照功能，可根据具体的情况而定。如果对照片保存的位置没什么要求，而且对照片的分辨率要求不高。可以使用系统提供的拍照功能，否则，就要自己来实现拍照功能了。

虽然到现在为止拍照的功能已经完全实现了，但程序在手机或模拟器上仍然不能正常运行，原因是需要在 AndroidManifest.xml 文件中设置拍照的权限许可（在调用系统提供的拍照功能时并不需要设置拍照权限许可），代码如下：

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

本例的运行效果与 14.3.1 节的例子的运行效果类似，只是在拍照时需要触摸屏幕才能结束拍照。

14.4　传感器在手机中的应用

自从苹果公司在 2007 年发布第一代 iPhone 以来，以前看似和手机挨不着边的传感器也逐渐成为手机硬件的重要组成部分。如果读者使用过 iPhone、HTC Dream、HTC Magic、HTC Hero 以及其他的 Android 手机，会发现通过将手机横向或纵向放置，屏幕会随着手机位置的不同而改变方向。这种功能就需要通过重力传感器来实现，除了重力传感器，还有很多其他类型的传感器被应用到手机中，例如磁阻传感器就是最重要的一种传感器。虽然手机可以通过 GPS 来判断方向，但在 GPS 信号不好或根本没有 GPS 信号的情况下，GPS 就形同虚设。这时通过磁阻传感器就可以很容易判断方向（东、南、西、北）。有了磁阻传感器，也使罗盘（俗称指向针）的电子化成为可能。

14.4.1　在应用程序中使用传感器

本节的例子代码所在的工程目录是 src\ch14\ch14_sensor

在 Android 应用程序中使用传感器要依赖于 android.hardware.SensorEventListener 接口。通过该接口可以监听传感器的各种事件。SensorEventListener 接口的代码如下：

```
package android.hardware;
public interface SensorEventListener
{
    public void onSensorChanged(SensorEvent event);
    public void onAccuracyChanged(Sensor sensor, int accuracy);
}
```

在 SensorEventListener 接口中定义了两个方法：onSensorChanged 和 onAccuracyChanged。当传感器的值发生变化时，例如磁阻传感器的方向改变时会调用 onSensorChanged 方法。当传感器的精度变化时会调用 onAccuracyChanged 方法。

onSensorChanged 方法只有一个 SensorEvent 类型的参数 event，其中 SensorEvent 类有一个 values 变量非常重要，该变量的类型是 float[]。但该变量最多只有 3 个元素，而且根据传感器的不同，values 变量中元素所代表的含义也不同。

在解释 values 变量中元素的含义之前，先来介绍一下 Android 的坐标系统是如何定义 X、Y、Z 轴的。

- X 轴的方向是沿着屏幕的水平方向从左向右。如果手机不是正方形的话，较短的边需要水平放置，较长的边需要垂直放置。
- Y 轴的方向是从屏幕的左下角开始沿着屏幕的垂直方向指向屏幕的顶端。
- 将手机平放在桌子上，Z 轴的方向是从手机里指向天空。

下面是 values 变量的元素在主要的传感器中所代表的含义。

1．方向传感器

在方向传感器中 values 变量的 3 个值都表示度数，它们的含义如下：

- values[0]：该值表示方位，也就是手机绕着 Z 轴旋转的角度。0 表示北（North）；90 表示东（East）；180 表示南（South）；270 表示西（West）。如果 values[0]的值正好是这 4 个值，并且手机是水平放置，表示手机的正前方就是这 4 个方向。可以利用这个特性来实现电子罗盘，实例 76 将详细介绍电子罗盘的实现过程。
- values[1]：该值表示倾斜度，或手机翘起的程度。当手机绕着 X 轴倾斜时该值发生变化。values[1]的取值范围是-180≤values[1] ≤180。假设将手机屏幕朝上水平放在桌子上，这时如果桌子是完全水平的，values[1]的值应该是 0（由于很少有桌子是绝对水平的，因此，该值很可能不为 0，但一般都是-5 和 5 之间的某个值）。这时从手机顶部开始抬起，直到将手机沿 X 轴旋转 180 度（屏幕向下水平放在桌面上）。在这个旋转过程中，values[1]会在 0 到-180 之间变化，也就是说，从手机顶部抬起时，values[1]的值会逐渐变小，直到等于-180。如果从手机底部开始抬起，直到将手机沿 X 轴旋转 180 度，这时 values[1]会在 0 到 180 之间变化。也就是 values[1]的值会逐渐增大，直到等于 180。可以利用 values[1]和下面要介绍的 values[2]来测量桌子等物体的倾斜度。
- values[2]：表示手机沿着 Y 轴的滚动角度。取值范围是-90≤values[2]≤90。假设将手机屏幕朝上水平放在桌面上，这时如果桌面是平的，values[2]的值应为 0。将手机左侧逐渐抬起时，values[2]的值逐渐变小，直到手机垂直于桌面放置，这时 values[2]的值是-90。将手机右侧逐渐抬起时，values[2]的值逐渐增大，直到手机垂直于桌面放置，这时 values[2]的值是 90。在垂直位置时继续向右或向左滚动，values[2]的值会继续在-90 至 90 之间变化。

2．加速传感器

该传感器的 values 变量的 3 个元素值分别表示 X、Y、Z 轴的加速值。例如，水平放在桌面上的手机从左侧向右侧移动，values[0]为负值；从右向左移动，values[0]为正值。读者可以通过本节的例子来体会加速传感器中的值的变化。

要想使用相应的传感器，仅实现 SensorEventListener 接口是不够的，还需要使用下面的代码来注册相应的传感器。

```
//  获得传感器管理器
SensorManager sm = (SensorManager) getSystemService(SENSOR_SERVICE);
//  注册方向传感器
sm.registerListener(this, sm.getDefaultSensor(Sensor.TYPE_ORIENTATION),
        SensorManager.SENSOR_DELAY_FASTEST);
```

如果想注册其他的传感器，可以改变 getDefaultSensor 方法的第 1 个参数值，例如，注册加速传感器可以使用 Sensor.TYPE_ACCELEROMETER。在 Sensor 类中还定义了很多传感器常量，但要根据手机中实际的硬件配置来注册传感器。如果手机中没有相应的传感器硬件，就算注册了相应的传感器也不起任何作用。getDefaultSensor 方法的第 2 个参数表示获得传感器数据的速度。SensorManager.SENSOR_DELAY_FASTEST 表示尽可能快地获得传感器数据。除了该值以外，还可以设置 3 个获得传感器数据的速度值，这些值如下：

- SensorManager.SENSOR_DELAY_NORMAL：默认的获得传感器数据的速度。
- SensorManager.SENSOR_DELAY_GAME：如果利用传感器开发游戏，建议使用该值。
- SensorManager.SENSOR_DELAY_UI：如果使用传感器更新 UI 中的数据，建议使用该值。

实例 76：电子罗盘

工程目录：src\ch14\ch14_compass

电子罗盘又叫电子指南针。在实现本例之前，先看一下如图 14.13 所示的运行效果。

图 14.13　电子罗盘（HTC Hero）

其中 N、S、W 和 E 分别表示北、南、西和东 4 个方向。

本例只使用了 onSensorChanged 事件方法及 values[0]。由于指南针图像上方是北，当手机前方是正北时（values[0]=0），图像不需要旋转。但如果不是正北，就需要将图像按一定角度旋转。假设当前 values[0]的值是 60，说明方向在东北方向。也就是说，手机顶部由北向东旋转。这时如果图像不旋转，N 的方向正好和正北的夹角是 60 度，需要将图像逆时针（从东向北旋转）旋转 60 度，N 才会指向正北方。因此，可以使用在 11.2.3 节介绍的旋转补间动画来旋转指南针图像，代码如下：

```
public void onSensorChanged(SensorEvent event)
{
    if (event.sensor.getType() == Sensor.TYPE_ORIENTATION)
    {
        float degree = event.values[0];
        //  以指南针图像中心为轴逆时针旋转 degree 度
        RotateAnimation ra = new RotateAnimation(currentDegree, -degree,
                Animation.RELATIVE_TO_SELF, 0.5f,
                Animation.RELATIVE_TO_SELF, 0.5f);
        //  在 200 毫秒之内完成旋转动作
        ra.setDuration(200);
        //  开始旋转图像
        imageView.startAnimation(ra);
        //  保存旋转后的度数，currentDegree 是一个在类中定义的 float 类型变量
        currentDegree = -degree;
    }
}
```

由于手机上带的一般都是二维磁阻传感器，因此应将手机放平才能使电子罗盘指向正确的方向。还要提一点的是，电子罗盘可能会受周围环境（例如，磁场）的影响而指向不正确的方向，读者在测试电子罗盘时应注意这一点。

实例 77：计步器

工程目录：src\ch14\ch14_stepcount

还可以利用方向传感器做出更有趣的应用，例如利用 values[1]或 values[2]的变化实现一个计步器。由于人在走路时会上下振动，因此，可以通过判断 values[1]或 values[2]中值的振荡变化进行计步。基本原理是在 onSensorChanged 方法中计算两次获得 values[1]值的差，并根据差值在一定范围之外开始计数，代码如下：

```
public void onSensorChanged(SensorEvent event)
{
    if (flag)
    {
        lastPoint = event.values[1];
        flag = false;
```

```
        }
        //  当两个 values[1]值之差的绝对值大于 8 时认为走了一步
        if (Math.abs(event.values[1] - lastPoint) > 8)
        {
            //  保存最后一步时的 values[1]的峰值
            lastPoint = event.values[1];
            //  将当前计数显示在 TextView 组件中
            textView.setText(String.valueOf(++count));
        }
    }
```

本例设置 3 个按钮用于控制计步的状态，这 3 个按钮可以控制开始计步、重值（将计步数清 0）和停止计步。这 3 个按钮的单击事件代码如下：

```
public void onClick(View view)
{
    String msg = "";
    switch (view.getId())
    {
        //  开始计步
        case R.id.btnStart:
            sm = (SensorManager) getSystemService(SENSOR_SERVICE);
            //  注册方向传感器
            sm.registerListener(this, sm
                    .getDefaultSensor(Sensor.TYPE_ORIENTATION),
                    SensorManager.SENSOR_DELAY_FASTEST);
            msg = "已经开始计步器.";
            break;
        //  重置计步器
        case R.id.btnReset:
            count = 0;
            msg = "已经重置计步器.";
            break;
        //  停止计步
        case R.id.btnStop:
            //  注销方向传感器
            sm.unregisterListener(this);
            count = 0;
            msg = "已经停止计步器.";
            break;
    }
    textView.setText(String.valueOf(count));
    Toast.makeText(this, msg, Toast.LENGTH_SHORT).show();
}
```

运行本例后，单击【开始】按钮，将手机放在兜里，再走两步看看，计步的效果如图 14.14 所示。

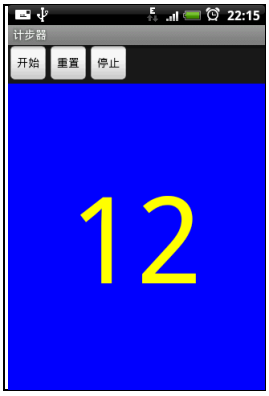


图 14.14　计步器（HTC Hero）

由于不同人走路的振动幅度不同，计步器并不会很准确地记录所走的步数。计步器只是一个有趣的应用而已，并不能用来准确计算所走的步数。如果想精确地统计所走的步数，最好的方法是在鞋底安装压力传感器，不过这就与本书的主题无关了。顺便提一下，方向和加速传感器经常用于实现游戏，例如，通过方向传感器可以控制游戏中飞机的飞行方向和速度。至于是否可以更精妙地运用各种传感器，就要靠读者的想象力了。

14.5　GPS 与地图定位

电子罗盘虽然可以指明方向，但却不能指向目前所在的具体位置，当然更不能指明行走路线。不过幸好现在的智能手机大多都提供了 GPS 模块，通过 GPS 模块可以接收 GPS 信号，并可精确地指定目前所在的位置（根据周围环境的不同，例如建筑物、天气、电磁干扰，GPS 的精确度会差很多，民用的 GPS 可能精度在 10 米至几百米之间，在空旷的地方使用 GPS 效果最好）。如果将 GPS 定位功能应用到地图上，还可以实现导航、搜索公交/驾车路线等有趣的功能。

14.5.1　Google 地图

本节的例子代码所在的工程目录是 src\ch14\ch14_map

Google 是以搜索引擎闻名，但 Google 不仅仅有搜索引擎。Google 著名的代码托管服务（http://code.google.com）包含了大量官方及第三方的优秀应用，其中 Android 就是最受关注的应用之一，除此之外，Google Maps API 也被大量应用在各种类型的场合。通过 Google Maps API 可以从 Google 下载地图，并通过经纬度或地点描述来确定具体的位置。本节的例子就是通过 Google 地图和 Google Maps API 来显示“沈阳三好街”的具体位置，并在该位置显示图像和文字作为标记，如图 14.15 所示。在 HTC Hero 手机上的运行效果如图 14.16 所示。

图 14.15　地图定位

图 14.16　地图定位（HTC Hero）

如果在手机上测试本节的例子，手机需要连接互联网。如果读者的手机已关闭互联网连接，请先打开互联网连接再运行程序。

下载和访问 Google 地图需要用到 com.google.android.maps.MapView 类，在开发基于 Google 地图的程序时需要使用支持 Google API 的 AVD 设备，在运行模拟器时也应启动支持 Google API 的模拟器。MapView

组件和其他组件的使用方法类似，只需要在 XML 布局文件中定义即可。但要想成功下载 Google 地图，还需要申请一个密钥。此密钥可以通过如下地址免费申请：

```
http://code.google.com/intl/zh-CN/android/maps-api-signup.html
```

在申请密钥之前，需要一个用于签名的密钥文件（详细介绍见 2.3 节的内容），一般以.keystore 结尾。如果用于开发程序，可以使用 debug.keystore 文件。该文件的路径如下：

```
C:\Documents and Settings\Administrator\.android\debug.keystore
```

单击 Eclipse 的【Window】>【Preferences】菜单项，打开【Preferences】对话框，在左侧找到 Android 节点，单击【Build】子节点，在右侧的【Build】设置页中的【Default debug keystore】文本框的值就是 debug.keystore 文件的路径。

在 Windows 控制台中进入 C:\Documents and Settings\Administrator\.android 目录，并输入如下命令：

```
keytool -list -keystore debug.keystore
```

当要求输入密码时，输入 android。按回车键后会在控制台中显示 debug.keystore 文件的认证指纹，如图 14.17 所示白色框中的内容。

获得认证指纹后，在申请密钥的页面下方选中同意协议复选框，并在【My certificate's MD5 fingerprint】文本框中输入认证指纹，单击【Generate API Key】按钮。如果输入的认证指纹是有效的，会产生一个新的页面，该页面中包含了申请的密钥。如图 14.18 所示黑色框中的内容。

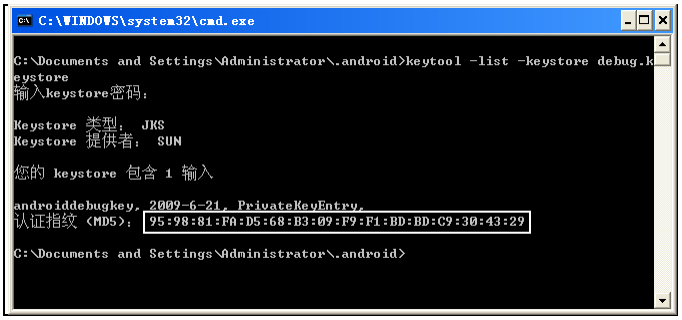


图 14.17　获得 debug.keystore 的认证指纹

图 14.18　成功获得密钥

在获得密钥后，需要在 XML 布局文件中定义 MapView 组件，并使用 android:apiKey 指定这个密钥，代码如下：

```
<com.google.android.maps.MapView
    android:id="@+id/mapview" android:layout_width="fill_parent"
    android:layout_height="fill_parent" android:apiKey="0H4t1kSw5lRhK_6D1GZMdfC_-KtCmEsVDB49Saw" />
```

在使用 MapView 组件时应注意如下 4 点：

- 由于 MapView 组件在 com.google.android.maps 包中，该包属于 Android SDK 附带的 jar 包（在<Android SDK 安装目录>\add-ons 目录中可以找到相应 Android SDK 版本的 jar 文件），并不属于 Android SDK 的一部分，因此，在定义 MapView 组件时必须带上包名。

- 虽然安装 ADT 都会产生一个 debug.keystore 文件，但在作者机器上的 debug.keystore 和读者机器上的 debug.keystore 是不一样的。读者在运行本例之前，应先使用自己机器上的 debug.keystore 文件获得认证指纹，并申请密钥，再用所获得的密钥替换 android:apiKey 属性的值。
- 如果读者要发布基于 Google Map 的应用程序，需要使用自己生成的 keystore 文件重新获得密钥，android:apiKey 的值应为新获得的密钥。
- 由于 MapView 需要访问互联网，因此，在 AndroidManifest.xml 文件中需要使用 android.permission.INTERNET 打开互联网访问权限。

在地图中定位需要使用经度和纬度。本例通过 Geocoder 类的 getFromLocationName 方法获得了“沈阳三好街”的准确位置（经纬度），并根据获得的经纬度创建了 GeoPoint 对象，以便在地图上定位。如果想在地图上添加其他元素，例如添加一个图像，可以使用 Overlay 对象。通过覆盖 Overlay 类的 draw 方法可以在地图上绘制任意的图形（包括文字和图像）。

下面先看一下使用 MapView 和在地图上定位的代码。

```
public void onCreate(Bundle savedInstanceState)
{
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    //  从 XML 布局文件获得 MapView 对象
    MapView mapView = (MapView) findViewById(R.id.mapview);
    //  允许通过触摸拖动地图
    mapView.setClickable(true);
    //  当触摸地图时在地图下方会出现缩放按钮，几秒后就会消失
    mapView.setBuiltInZoomControls(true);
    //  获得 MapController 对象，mapController 是一个在类中定义的 MapController 类型变量
    mapController = mapView.getController();
    //  创建 Geocoder 对象，用于获得指定地点的地址
    Geocoder gc = new Geocoder(this);
    //  将地图设为 Traffic 模式
    mapView.setTraffic(true);
    try
    {
        //  查询指定地点的地址
        List<Address> addresses = gc.getFromLocationName("沈阳三好街", 5);
        //  根据经纬度创建 GeoPoint 对象
        geoPoint = new GeoPoint(
                (int) (addresses.get(0).getLatitude() * 1E6),
                (int) (addresses.get(0).getLongitude() * 1E6));
        setTitle(addresses.get(0).getFeatureName());
    }
    catch (Exception e)
    {
    }
    //  创建 MyOverlay 对象，用于在地图上绘制图形
    MyOverlay myOverlay = new MyOverlay();
    //  将 MyOverlay 对象添加到 MapView 组件中
    mapView.getOverlays().add(myOverlay);
    //  设置地图的初始大小，范围在 1 和 21 之间。1：最小尺寸，21：最大尺寸
    mapController.setZoom(20);
    //  以动画方式进行定位
    mapController.animateTo(geoPoint);
}
//  用于在地图上绘制图形的 MyOverlay 对象
class MyOverlay extends Overlay
{
    @Override
    public boolean draw(Canvas canvas, MapView mapView, boolean shadow, long when)
    {
        Paint paint = new Paint();
```

```
            paint.setColor(Color.RED);
            Point screenPoint = new Point();
            //  将“沈阳三好街”在地图上的位置转换成屏幕的实际坐标
            mapView.getProjection().toPixels(geoPoint, screenPoint);
            Bitmap bmp = BitmapFactory.decodeResource(getResources(), R.drawable.flag);
            //  在地图上绘制图像
            canvas.drawBitmap(bmp, screenPoint.x, screenPoint.y, paint);
            //  在地图上绘制文字
            canvas.drawText("三好街",screenPoint.x, screenPoint.y, paint);
            return super.draw(canvas, mapView, shadow, when);
        }
    }
```

在编写上面代码时应注意如下 3 点：

- MapView 有 3 种地图模式：交通（Traffic）、卫星（Satellite）和街景（StreetView）。在上面的代码中使用 setTraffic 方法将地图设成交通地图模式，还可以通过 setSatellite 和 setStreetView 方法将地图设成卫星和街景视图。例如图 14.19 是“沈阳三好街”的卫星地图（地图并不是实时的，会与当前的卫星照片有一定的偏差）。
- Geocoder 类的 getFromLocationName 方法会返回所有满足地点描述的地址（List 对象）。该方法的第 1 个参数表示地点描述，第 2 个参数表示最多返回的地址。如果满足地点描述的地点多于第 2 个参数所指定的值，getFromLocationName 方法会只返回第 2 个参数指定的地址数。如果少于指定的返回地址数，则返回实际的地址数。
- 由于 GeoPoint 类的构造方法需要将经纬度扩大 100 万倍，因此，通过 getLongitude 和 getLatitude 方法获得的经纬度需要乘以 1000000 才可以被 GeoPoint 类使用，用科学计数法表示就是 1E6。

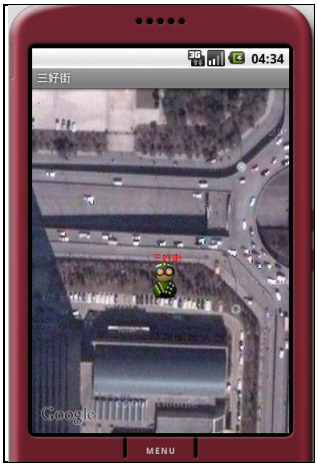


图 14.19　“沈阳三好街”的卫星地图

14.5.2　用 GPS 定位到当前位置

本节的例子代码所在的工程目录是 src\ch14\ch14_gps

要想定位到当前的位置，需要利用手机中的 GPS 模块。使用 GPS 首先需要获得 LocationManager 服务，代码如下：

```
LocationManager locationManager = (LocationManager) getSystemService(Context.LOCATION_SERVICE);
```

通过 LocationManager 类的 getBestProvider 方法可以获得当前的位置，但需要通过 Criteria 对象指定一些参数，代码如下：

```
Criteria criteria = new Criteria();
//  获得最好的定位效果
criteria.setAccuracy(Criteria.ACCURACY_FINE);
criteria.setAltitudeRequired(false);
criteria.setBearingRequired(false);
```

```
criteria.setCostAllowed(false);
//  使用省电模式
criteria.setPowerRequirement(Criteria.POWER_LOW);
//  获得当前的位置提供者
String provider = locationManager.getBestProvider(criteria, true);
//  获得当前的位置
Location location = locationManager.getLastKnownLocation(provider);
//  获得当前位置的纬度
Double latitude = location.getLatitude() * 1E6;
//  获得当前位置的经度
Double longitude = location.getLongitude() * 1E6;
```

在获得当前位置的经纬度后，剩下的工作就和 14.5.1 节的例子一样了。只是在本例中还输出了与当前位置相关的信息，代码如下：

```
Geocoder gc = new Geocoder(this);
List<Address> addresses = gc.getFromLocation(location.getLatitude(), location.getLongitude(), 1);
if (addresses.size() > 0)
{
    msg += "AddressLine：" + addresses.get(0).getAddressLine(0)+ "\n";
    msg += "CountryName：" + addresses.get(0).getCountryName()+ "\n";
    msg += "Locality：" + addresses.get(0).getLocality() + "\n";
    msg += "FeatureName：" + addresses.get(0).getFeatureName();
}
textView.setText(msg);
```

本例需要使用如下代码在 AndroidManifest.xml 文件中打开相应的权限：

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_FIND_LOCATION" />
```

在手机上运行本节的例子，会显示如图 14.20 所示的效果。

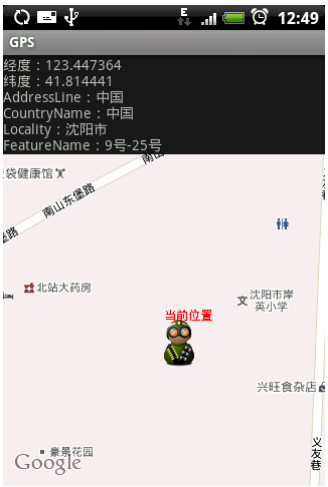


图 14.20　定位到当前位置（HTC Hero）

14.6　WIFI

图 14.21　WIFI（HTC Hero）

工程目录：src\ch14\ch14_wifi

WIFI 的全称是 Wireless Fidelity，又称 802.11b 标准，是一种高速的无线通信协议，传输速度可以达到 11Mb/s。实际上，对 WIFI 并不需要过多的控制（当成功连接 WIFI 后，就可以直接通过 IP 在 WIFI 设备之间进行通信了），一般只需要控制打开或关闭 WIFI 以及获得一些与 WIFI 相关的信息（例如，MAC 地址、IP 等）。如果读者的 Android 手机有 WIFI 功能，可以

在手机上测试本节的例子。要注意的是，WIFI 功能不能在 Android 模拟器上测试，就算在有 WIFI 功能的真机上也需要先通过 WIFI 和计算机或其他 WIFI 设备连接后，才能获得与 WIFI 相关的信息。

本节的例子可以关闭和开始 WIFI，并获得各种与 WIFI 相关的信息。首先确认手机通过 WIFI 与其他 WIFI 设备成功连接，然后运行本节的例子，会看到如图 14.21 所示的输出信息。

本例的完整实现代码如下：

```
package net.blogjava.mobile.wifi;

import java.net.Inet4Address;
import java.util.List;
import android.app.Activity;
import android.content.Context;
import android.net.wifi.WifiConfiguration;
import android.net.wifi.WifiInfo;
import android.net.wifi.WifiManager;
import android.os.Bundle;
import android.widget.CheckBox;
import android.widget.CompoundButton;
import android.widget.TextView;
import android.widget.CompoundButton.OnCheckedChangeListener;

public class Main extends Activity implements OnCheckedChangeListener
{
    private WifiManager wifiManager;
    private WifiInfo wifiInfo;
    private CheckBox chkOpenCloseWifiBox;
    private List<WifiConfiguration> wifiConfigurations;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        //  获得 WifiManager 对象
        wifiManager = (WifiManager) getSystemService(Context.WIFI_SERVICE);
        //  获得连接信息对象
        wifiInfo = wifiManager.getConnectionInfo();
        chkOpenCloseWifiBox = (CheckBox) findViewById(R.id.chkOpenCloseWifi);
        TextView tvWifiConfigurations = (TextView) findViewById(R.id.tvWifiConfigurations);
        TextView tvWifiInfo = (TextView) findViewById(R.id.tvWifiInfo);
        chkOpenCloseWifiBox.setOnCheckedChangeListener(this);
        //  根据当前 WIFI 的状态（是否被打开）设置复选框的选中状态
        if (wifiManager.isWifiEnabled())
        {
            chkOpenCloseWifiBox.setText("Wifi 已开启");
            chkOpenCloseWifiBox.setChecked(true);
        }
        else
        {
            chkOpenCloseWifiBox.setText("Wifi 已关闭");
            chkOpenCloseWifiBox.setChecked(false);
        }

        // 获得 WIFI 信息
        StringBuffer sb = new StringBuffer();
        sb.append("Wifi 信息\n");
        sb.append("MAC 地址：" + wifiInfo.getMacAddress() + "\n");
        sb.append("接入点的 BSSID：" + wifiInfo.getBSSID() + "\n");

        sb.append("IP 地址（int）：" + wifiInfo.getIpAddress() + "\n");
        sb.append("IP 地址（Hex）：" + Integer.toHexString(wifiInfo.getIpAddress()) + "\n");
        sb.append("IP 地址：" + ipIntToString(wifiInfo.getIpAddress()) + "\n");
        sb.append("网络 ID：" + wifiInfo.getNetworkId() + "\n");
```

```
            tvWifiInfo.setText(sb.toString());

            // 得到配置好的网络
            wifiConfigurations = wifiManager.getConfiguredNetworks();
            tvWifiConfigurations.setText("已连接的无线网络\n");
            for (WifiConfiguration wifiConfiguration : wifiConfigurations)
            {
                tvWifiConfigurations.setText(tvWifiConfigurations.getText() + wifiConfiguration.SSID + "\n");
            }
        }
        //  将 int 类型的 IP 转换成字符串形式的 IP
        private String ipIntToString(int ip)
        {
            try
            {
                byte[] bytes = new byte[4];
                bytes[0] = (byte) (0xff & ip);
                bytes[1] = (byte) ((0xff00 & ip) >> 8);
                bytes[2] = (byte) ((0xff0000 & ip) >> 16);
                bytes[3] = (byte) ((0xff000000 & ip) >> 24);
                return Inet4Address.getByAddress(bytes).getHostAddress();
            }
            catch (Exception e)
            {
                return "";
            }
        }
        @Override
        public void onCheckedChanged(CompoundButton buttonView, boolean isChecked)
        {
            //  当选中复选框时打开 WIFI
            if (isChecked)
            {
                wifiManager.setWifiEnabled(true);
                chkOpenCloseWifiBox.setText("Wifi 已开启");
            }
            //  当取消复选框选中状态时关闭 WIFI
            else
            {
                wifiManager.setWifiEnabled(false);
                chkOpenCloseWifiBox.setText("Wifi 已关闭");
            }
        }
    }
```

在 AndroidManifest.xml 文件中要使用如下的代码打开相应的权限。

```
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE"></uses-permission>
<uses-permission android:name="android.permission.WAKE_LOCK"></uses-permission>
<uses-permission android:name="android.permission.CHANGE_WIFI_STATE"></uses-permission>
```

14.7　本章小结

本章主要介绍了如何在手机上测试和调试需要使用手机硬件的应用程序。这些硬件包括麦克风、摄像头、传感器、GPS 和 WIFI。这些硬件都是智能手机标准的配置，尤其是近年来传感器被大量用于手机中。除了本章介绍的方向传感器和加速传感器外，还包括光学传感器、温度传感器、压力传感器在内的多种传感器被应用在以手机为主的移动设备中。在未来，传感器及其他先进的电子设备将成为智能手机的一部分，而手机拥有了这些设备，就不再只是手机了，而会成为无所不能的智能终端，真正实现 All In One 的时代已为时不远了。