19

有趣的 Android 技术

本章将介绍一些有趣的 Android 技术，例如可以通过手势进行输入、TTS 语音朗读，以及从 Android 2.1 开始支持的动态壁纸技术。

本章内容

- 创建手势文件
- 用手势输入文本
- 用手势调用应用程序
- 编写自己的手势创建器
- 用 TTS 朗读文本
- 动态壁纸

19.1 手势（Gesture）

看到“手势”这个词，千万不要以为是像哑语一样的动作手势。实际上，这里的手势就是指手写输入，只是叫“手势”更形象些。在手机中经常会使用手写输入，这就是所谓的手势。本节要介绍的手势与手写输入类似，但不同的是手写输入一次只能输入一个汉字或字母，而本节要介绍的每个手势可以对应一个字符串，也就是说，通过在手机屏幕上画一个手势，可以直接输入一个字符串。除此之外，还可以将某个手势与指定的应用程序相关联，例如通过手势可以拨打电话。

19.1.1 创建手势文件

在使用手势之前，需要建立一个手势文件。在识别手势时，需要装载这个手势文件，并通过手势文件中的描述来识别手势。

从 Android 1.6 开始，发行包中都带了一个 GestureBuilder 工程，该工程可用来建立手势文件。读者可以在<Android SDK 安装目录>\platforms\android-1.6\samples 目录中找到该工程。如果读者使用的是其他 Android 版本，需要将 android-1.6 改成其他的名字，例如 android-2.0。

在模拟器上安装并运行该工程生成的 apk 文件，会显示如图 19.1 所示的界面。单击【Add gesture】按钮增加一个手势。在增加手势界面上方的文本框中输入一个手势名（在识别手势后，系统会返回该名称），并在下方的空白处随意画一些手势轨迹，如图 19.2 所示。需要注意的是，系统允许多个手势对应于同一个手势名。读者可以采用同样的方法多增加几个手势。在创建完手势后，读者会看到 SD 卡的根目录多了个 gestures 文件，该文件是二进制格式。在 19.1.2 节将看到如何使用刚创建的手势文件来识别手势。

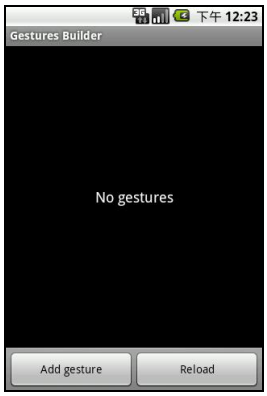

图 19.1 手势创建器的主界面

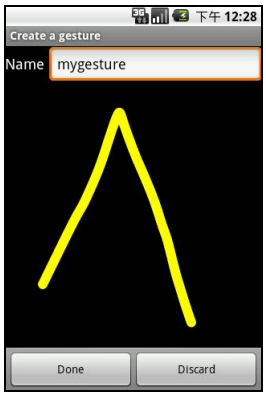

图 19.2 增加一个手势

19.1.2 通过手势输入字符串

工程目录：src\ch19\ch19_gesture_text。

手势的一个重要应用就是在屏幕上简单地画几笔就可以输入复杂的内容。本节的例子会使用在 19.1.1 节介绍的 GestureBuilder 程序建立 3 个手势，如图 19.3 所示。运行本例后，在屏幕上画如图 19.4 所示的图形，系统会匹配如图 19.3 所示的 3 个手势中的第 1 个。松开鼠标后，会将识别后的信息以 Toast 信息提示框的形式显示，如图 19.5 所示。读者也可以将这些信息插入到 EditText 或其他的组件中。

在匹配信息中有一个 score 字段，该字段表示匹配的程度。一般该字段的值大于 1，就认为可能与手势匹配。如果有多个手势可能匹配我们绘制的手势，可以提供一个选择列表，以便用户

可以准确地选择匹配结果。这有些像手写输入，有很多时候都会出现一个可能匹配的列表，最终由用户决定哪个是最终的匹配结果。

图 19.3　建立的 3 个手势

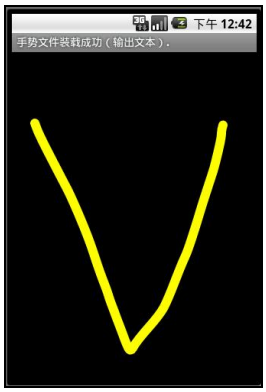

图 19.4　画手势

图 19.5　显示匹配的信息

在如图 19.4 所示的界面中绘制手势的组件是 android.gesture.GestureOverlayView。该组件不是标准的 Android 组件，因此在 XML 布局文件中定义该组件时必须使用全名（包名+类名）。

```
<android.gesture.GestureOverlayView
    android:id="@+id/gestures" android:layout_width="fill_parent"
    android:layout_height="fill_parent" android:gestureStrokeType="multiple" />
```

其中 android:gestureStrokeType 属性表示 GestureOverlayView 组件是否可接受多个手势数。也就是说，一个完整的手势可能由多个不连续的图形组成，例如乘号由两个斜线组成。如果将该属性值设为 multiple，表示可以绘制由多个不连续图形组成的手势；如果将该属性值设为 single，绘制手势时就只能使用一笔画了（中间不能断），这有些像手写输入。对于大部分汉字来说，都是由不连续的笔画组成的（连笔字除外），这就需要由多个手势来绘制一个汉字。

下面来装载手势文件。本例将手势文件放在 res\raw 目录中，也可以将手势文件放在 SD 卡或手机内存中。装载手势文件的代码如下：

```
//  指定手势资源文件的位置
gestureLibrary = GestureLibraries.fromRawResource(this, R.raw.gestures);
//  从 raw 资源中装载手势资源
if (gestureLibrary.load())
{
    setTitle("手势文件装载成功（输出文本）.");
    GestureOverlayView gestureOverlayView = (GestureOverlayView) findViewById(R.id.gestures);
    //  设置 OnGesturePerformedListener 事件，该事件方法在绘制完手势并进行识别后调用
    gestureOverlayView.addOnGesturePerformedListener(this);
```

```
}
else
{
    setTitle("手势文件装载失败.");
}
```

其中 gestureLibrary 是在类中定义的 android.gesture.GestureLibrary 类型变量。在成功装载手势资源后，需要为 GestureOverlayView 组件指定 OnGesturePerformedListener 事件，该事件方法的代码如下：

```
public void onGesturePerformed(GestureOverlayView overlay, Gesture gesture)
{
    //  获得可能匹配的手势
    ArrayList<Prediction> predictions = gestureLibrary.recognize(gesture);
    //  有可能匹配的手势
    if (predictions.size() > 0)
    {
        StringBuilder sb = new StringBuilder();
        int n = 0;
        //  开始扫描所有可能匹配的手势
        for (int i = 0; i < predictions.size(); i++)
        {
            Prediction prediction = predictions.get(i);
            // 根据相似度，只列出 score 字段值大于 1 的匹配手势
            if (prediction.score > 1.0)
            {
                sb.append("score:" + prediction.score + "    name:"
                        + prediction.name + "\n");
                n++;
            }
        }
        sb.insert(0,n + "个相匹配的手势.\n");
        //  显示最终的匹配信息
        Toast.makeText(this, sb.toString(), Toast.LENGTH_SHORT).show();
    }
}
```

需要注意的是，手势采用了相似度进行匹配。这就意味着预设的手势越多，手势的图形越相似，与同一个绘制的手势匹配的结果就可能越多。score 字段可以认为是相似度（指绘制的手势和手势库中手势的相似性），一般取相似度大于 1 的手势即可。当然，如果要求更精确，也可以提高相似度。

19.1.3 通过手势调用程序

工程目录：src\ch19\ch19_gesture_action。

只要在 onGesturePerformed 方法中获得手势名，并按照一定规则就可以调用其他的应用程序。本例通过 3 个手势来拨打电话、显示通话记录和自动输入电话号，这 3 个手势如图 19.6 所示。

图 19.6　调用程序的 3 个手势

通过这 3 个手势返回的 action_call、action_call_button 和 action_dial 来决定调用哪个程序，代码如下：

```
public void onGesturePerformed(GestureOverlayView overlay, Gesture gesture)
{
    ArrayList<Prediction> predictions = gestureLibrary.recognize(gesture);
    if (predictions.size() > 0)
    {
        int n = 0;
        for (int i = 0; i < predictions.size(); i++)
        {
            Prediction prediction = predictions.get(i);
            if (prediction.score > 1.0)
            {
                Intent intent = null;
                Toast.makeText(this, prediction.name, Toast.LENGTH_SHORT).show();
                if ("action_call".equals(prediction.name))
                {
                    //  拨打电话
                    intent = new Intent(Intent.ACTION_CALL, Uri.parse("tel:12345678"));
                }
                else if ("action_call_button".equals(prediction.name))
                {
                    //  显示通话记录
                    intent = new Intent(Intent.ACTION_CALL_BUTTON);
                }
                else if ("action_dial".equals(prediction.name))
                {
                    //  将电话传入拨号程序
                    intent = new Intent(Intent.ACTION_DIAL, Uri.parse("tel:12345678"));
                }
                if (intent != null)
                    startActivity(intent);
                n++;
```

```
                    break;
                }
            }
            if (n == 0)
                Toast.makeText(this, "没有符合要求的手势.", Toast.LENGTH_SHORT).show();
        }
    }
```

19.1.4 编写自己的手势创建器

工程目录：src\ch19\ch19_gesture_builder。

有时候需要在自己的程序中加入创建手势的功能。本节就来学习一下建立手势文件的原理，感兴趣的读者也可以去分析 GestureBuilder 工程中的源代码，但本例更直接地描述了手势创建器的编写过程。

创建手势需要 GestureOverlayView 组件的另外一个事件：OnGestureListener。该事件需要指定一个对象。在开始绘制手势、绘制的过程、绘制结束以及取消绘制时都会调用该事件对象中的方法。指定 OnGestureListener 事件的代码如下：

```
GestureOverlayView overlay = (GestureOverlayView) findViewById(R.id.gestures_overlay);
overlay.addOnGestureListener(new GesturesProcessor());
```

其中 GesturesProcessor 是一个事件类，代码如下：

```
private class GesturesProcessor implements GestureOverlayView.OnGestureListener
{
    public void onGestureStarted(GestureOverlayView overlay, MotionEvent event)
    {
    }
    public void onGesture(GestureOverlayView overlay, MotionEvent event)
    {
    }
    public void onGestureEnded(final GestureOverlayView overlay, MotionEvent event)
    {
        final Gesture gesture = overlay.getGesture();
        View gestureView = getLayoutInflater().inflate(R.layout.gesture, null);
        final TextView textView = (TextView) gestureView.findViewById(R.id.textview);
        ImageView imageView = (ImageView) gestureView.findViewById(R.id.imageview);
        //  获得绘制的手势的图像（128*128），0xFFFFFF00 表示图像中手势的颜色（黄色）
        Bitmap bitmap = gesture.toBitmap(128, 128, 8, 0xFFFFFF00);
        //  在 ImageView 组件中显示手势图形
        imageView.setImageBitmap(bitmap);
        textView.setText("手势名： " + editText.getText());
        new AlertDialog.Builder(Main.this).setView(gestureView)
                .setPositiveButton("保存", new OnClickListener()
                {
                    @Override
                    public void onClick(DialogInterface dialog, int which)
```

```
                    {
                        GestureLibrary store = GestureLibraries.fromFile("/sdcard/mygestures");
                        store.addGesture(textView.getText().toString(), gesture);
                        //  保存手势文件
                        store.save();
                    }
                }).setNegativeButton("取消", null).show();
        }
        public void onGestureCancelled(GestureOverlayView overlay, MotionEvent event)
        {
        }
    }
```

在 GesturesProcessor 类中有 4 个事件方法，但只使用了 onGestureEnded 方法。当绘制完手势后，会调用该方法。创建手势文件的基本原理是通过 Gesture 类的 toBitmap 方法获得绘制手势的 Bitmap 对象，然后将其显示在 ImageView 中，并在 TextView 中显示手势名，将这两个组件显示在一个对话框中。在绘制完手势后，会显示这个对话框，如图 19.7 所示。如果确定手势和手势名无误，则单击【保存】按钮创建手势文件（如果存在则打开手势文件），并保存当前手势和手势名。读者可以在 SD 卡的根目录中找到保存手势的 mygestures 文件。

图 19.7　保存手势

从 Android 1.6 开始，在默认的情况下不允许向 SD 卡写数据。要想写入数据，需要使用<uses-permission>标签设置 android.permission.WRITE_EXTERNAL_STORAGE 权限。如果读者的程序中需要向 SD 卡写数据，并且以前是用 Android 1.5 开发的，而将来需要在 Android 的更高版本中运行，建议现在就使用<uses-permission>标签打开这个权限，否则程序将在 Android 1.6 以上的版本中无法成功向 SD 卡写数据。由于本例至少需要 Android 1.6 才能运行，因此也需要设置该权限，否则无法在 SD 卡的根目录生成 mygestures 文件。

19.2 让手机说话（TTS）

工程目录：src\ch19\ch19_tts。

方便输入信息还不够，如果让手机根据文本读出输入的内容那岂不是更人性化了。在 Android 1.6 中提供了 TTS（Text To Speech）技术可以完成这个工作。

TTS 技术的核心是 android.speech.tts.TextToSpeech 类。要想使用 TTS 技术朗读文本，需要做两个工作：初始化 TTS 和指定要朗读的文本。在第 1 项工作中主要指定 TTS 朗读的文本的语言，第 2 项工作主要使用 speak 方法指定要朗读的文本。

初始化 TTS 需要在 onInit 事件方法中完成。要使用该事件方法需要实现 TextToSpeech.OnInitListener 接口，在本例中当前类（Main 类）实现了该接口。创建 TextToSpeech 对象的代码如下：

```
//  tts 是 TextToSpeech 类型的对象，构造方法的第 1 个参数是 Context 类型的值，第 2 个参数需要
//  指定 TextToSpeech.OnInitListener 对象实例
tts = new TextToSpeech(this, this);
```

初始化 TTS 的代码如下：

```
public void onInit(int status)
{
    if (status == TextToSpeech.SUCCESS)
    {
        //  指定当前朗读的语言是英文
        int result = tts.setLanguage(Locale.US);
        if (result == TextToSpeech.LANG_MISSING_DATA
                || result == TextToSpeech.LANG_NOT_SUPPORTED)
        {
            Toast.makeText(this, "Language is not available.", Toast.LENGTH_SHORT).show();
        }
    }
}
```

下面的代码使用 speak 方法朗读了文本。

```
public void onClick(View view)
{
    tts.speak(textView.getText().toString(), TextToSpeech.QUEUE_FLUSH, null);
}
```

其中 speak 方法的第 1 个参数表示要朗读的文本。运行本例，单击【说话】按钮会朗读按钮下方的文字，如图 19.8 所示。

目前 TTS 只支持以英语为首的几种欧美语言，中文、日文等亚洲语言暂不支持。

图 19.8　朗读文本

19.3　动态壁纸

工程目录：src\ch19\ch19_livewallpapers。

动态壁纸的最低版本要求是 Android 2.1。

在手机桌面上放一张漂亮的图像是一件非常酷的事情。不过，这还不够酷。如果可以触摸桌面的空白处，会随着触摸的位置不同而发生各种变化，那岂不是更棒了。如果大家都是这么认为的，那么 Android 2.1 会成为目前 Android 中最“帅”的版本，因为 Android 2.1 提供了可以不断变化的动态壁纸，中文版的 Android 模拟器将其翻译成“当前壁纸”，不过叫“动态壁纸”会更贴切一些。

也许很多读者还不清楚什么是动态壁纸。那么现在先来看一下本节实现的例子。当触摸屏幕的任何空白位置时，会显示一个彩色的实心圆（颜色是随机变化的），如图 19.9 所示。要使用动态壁纸，需要在 Android 桌面的选项菜单中单击【壁纸】菜单项，在弹出的子菜单中选择【当前壁纸】菜单项，会显示如图 19.10 所示的界面。在该界面中可以预览动态壁纸的效果。当触摸界面的空白处时也会出现不同颜色的实心圆。单击【设置】按钮可以进入动态壁纸的设置页面，如图 19.11 所示。单击【配置圆的半径】配置项，会看到弹出如图 19.12 所示的配置项列表。读者可以选择各种大小的圆。

图 19.9　动态壁纸的效果

图 19.10　动态壁纸的预览界面

图 19.11　动态壁纸的设置界面

图 19.12　设置动态壁纸绘制的彩色实心圆的大小

动态壁纸的核心是一个服务类，该类必须是 android.service.wallpaper.WallpaperService 的子类。本例的服务类是 LiveWallpaperService，在该类中定义了一个 WallPaperEngine 类，该类是 WallpaperService.Engine 的子类，用于处理动态壁纸的核心业务。LiveWallpaperService 类的代码如下：

```
package net.blogjava.mobile.livewallpapers;

import android.content.SharedPreferences;
import android.service.wallpaper.WallpaperService;
```

```
import android.view.MotionEvent;
import android.view.SurfaceHolder;

public class LiveWallpaperService extends WallpaperService
{
    public static final String PREFERENCES = "net.blogjava.mobile.livewallpapers";
    public static final String PREFERENCE_RADIUS = "preference_radius";
    @Override
    public Engine onCreateEngine()
    {
        return new WallPaperEngine();                    //  创建动态壁纸引擎
    }
    //  定义动态壁纸引擎类
    public class WallPaperEngine extends Engine implements
            SharedPreferences.OnSharedPreferenceChangeListener
    {
        private LiveWallpaperPainting painting;
        private SharedPreferences prefs;
        //  在构造方法中需要读取配置文件中的信息，以确定绘制的彩色实心圆的半径
        public WallPaperEngine()
        {
            SurfaceHolder holder = getSurfaceHolder();
            prefs = LiveWallpaperService.this.getSharedPreferences(PREFERENCES, 0);
            prefs.registerOnSharedPreferenceChangeListener(this);
            painting = new LiveWallpaperPainting(holder,
                    getApplicationContext(), Integer.parseInt(prefs.getString(
                            PREFERENCE_RADIUS, "10")));
        }
        public void onSharedPreferenceChanged(SharedPreferences prefs, String key)
        {
            //  当设置变化时改变实心圆的半径
            painting.setRadius(Integer.parseInt(prefs.getString(PREFERENCE_RADIUS, "10")));
        }
        @Override
        public void onCreate(SurfaceHolder surfaceHolder)
        {
            super.onCreate(surfaceHolder);
            setTouchEventsEnabled(true);
        }
        @Override
        public void onDestroy()
        {
            super.onDestroy();
            painting.stopPainting();
        }

        @Override
        public void onVisibilityChanged(boolean visible)
        {
            if (visible)
```

```
            {
                painting.resumePainting();
            }
            else
            {
                painting.pausePainting();
            }
        }
        @Override
        public void onSurfaceChanged(SurfaceHolder holder, int format, int width, int height)
        {
            super.onSurfaceChanged(holder, format, width, height);
            painting.setSurfaceSize(width, height);
        }
        @Override
        public void onSurfaceCreated(SurfaceHolder holder)
        {
            super.onSurfaceCreated(holder);
            //  当 surface（绘制动态壁纸的界面）创建后，开始绘制彩色实心圆
            painting.start();
        }
        //  当 Surface 销毁时需要停止绘制壁纸
        @Override
        public void onSurfaceDestroyed(SurfaceHolder holder)
        {
            super.onSurfaceDestroyed(holder);
            boolean retry = true;
            painting.stopPainting();
            while (retry)
            {
                try
                {
                    painting.join();
                    retry = false;
                }
                catch (InterruptedException e)
                {
                }
            }
        }
        @Override
        public void onTouchEvent(MotionEvent event)
        {
            super.onTouchEvent(event);
            painting.doTouchEvent(event);
        }
    }
}
```

在上面的代码中涉及到一个 LiveWallpaperPainting 类，该类通过线程不断扫描用户在屏幕上触

摸的点，然后根据触摸点绘制彩色实心圆。该类的代码如下：

```
package net.blogjava.mobile.livewallpapers;

import java.util.ArrayList;
import java.util.List;
import java.util.Random;
import android.content.Context;
import android.graphics.Canvas;
import android.graphics.Paint;
import android.graphics.drawable.BitmapDrawable;
import android.view.MotionEvent;
import android.view.SurfaceHolder;

public class LiveWallpaperPainting extends Thread
{

    private SurfaceHolder surfaceHolder;
    private Context context;
    private boolean wait;
    private boolean run;
    /* 尺寸和半径  */
    private int width;
    private int height;
    private int radius;
    /* 触摸点 */
    private List<TouchPoint> points;
    /* 时间轨迹 */
    private long previousTime;
    public LiveWallpaperPainting(SurfaceHolder surfaceHolder, Context context, int radius)
    {
        this.surfaceHolder = surfaceHolder;
        this.context = context;
        // 直到 surface 被创建和显示时才开始动画
        this.wait = true;
        // 初始化触摸点
        this.points = new ArrayList<TouchPoint>();
        // 初始化半径
        this.radius = radius;
    }
    //  通过设置页面可以改变圆的半径
    public void setRadius(int radius)
    {
        this.radius = radius;
    }
    // 暂停动态壁纸的动画
    public void pausePainting()
    {
        this.wait = true;
        synchronized (this)
```

```
        {
            this.notify();
        }
    }
    //  恢复在动态壁纸上绘制彩色实心圆
    public void resumePainting()
    {
        this.wait = false;
        synchronized (this)
        {
            this.notify();
        }
    }
    //  停止在动态壁纸上绘制彩色实心圆
    public void stopPainting()
    {
        this.run = false;
        synchronized (this)
        {
            this.notify();
        }
    }
    @Override
    public void run()
    {
        this.run = true;
        Canvas canvas = null;
        while (run)
        {
            try
            {
                canvas = this.surfaceHolder.lockCanvas(null);
                synchronized (this.surfaceHolder)
                {
                    //  绘制彩色实心圆和背景图
                    doDraw(canvas);
                }
            } finally
            {
                if (canvas != null)
                {
                    this.surfaceHolder.unlockCanvasAndPost(canvas);
                }
            }
            // 如果不需要动画则暂停动画
            synchronized (this)
            {
                if (wait)
                {
                    try
```

```
                        {
                            wait();
                        }
                        catch (Exception e)
                        {
                        }
                    }
                }
            }
        }
        public void setSurfaceSize(int width, int height)
        {
            this.width = width;
            this.height = height;
            synchronized (this)
            {
                this.notify();
            }
        }
        public void doTouchEvent(MotionEvent event)
        {
            synchronized (this.points)
            {
                int color = new Random().nextInt(Integer.MAX_VALUE);
                //  将用户触摸屏幕的点信息保存在 points 中，以便在 run 方法中扫描这些点并绘制彩色实心圆
                points.add(new TouchPoint((int) event.getX(), (int) event.getY(),
                        color, Math.min(width, height) / this.radius));
            }
            this.wait = false;
            synchronized (this)
            {
                notify();
            }
        }
        private void doDraw(Canvas canvas)
        {
            long currentTime = System.currentTimeMillis();
            long elapsed = currentTime - previousTime;
            if (elapsed > 20)
            {
                BitmapDrawable bitmapDrawable =
                    (BitmapDrawable) context.getResources().getDrawable(R.drawable.background);
                //  绘制动态壁纸的背景图
                canvas.drawBitmap(bitmapDrawable.getBitmap(), 0, 0, new Paint());
                // 绘制触摸点
                Paint paint = new Paint();
                List<TouchPoint> pointsToRemove = new ArrayList<TouchPoint>();
                synchronized (this.points)
                {
                    for (TouchPoint point : points)
```

```
                {
                    paint.setColor(point.color);
                    point.radius -= elapsed / 20;
                    if (point.radius <= 0)
                    {
                        pointsToRemove.add(point);
                    }
                    else
                    {
                        canvas.drawCircle(point.x, point.y, point.radius,paint);
                    }
                }
                points.removeAll(pointsToRemove);
            }
            previousTime = currentTime;
            if (points.size() == 0)
            {
                wait = true;
            }
        }
    }
    //  保存绘制的彩色实心圆的信息
    class TouchPoint
    {
        int x;
        int y;
        int color;
        int radius;
        public TouchPoint(int x, int y, int color, int radius)
        {
            this.x = x;
            this.y = y;
            this.radius = radius;
            this.color = color;
        }
    }
}
```

下面来编写最后一个类（LiveWallpaperSettings），该类用于设置彩色实心圆的半径，代码如下：

```
package net.blogjava.mobile.livewallpapers;

import android.content.SharedPreferences;
import android.os.Bundle;
import android.preference.PreferenceActivity;

public class LiveWallpaperSettings extends PreferenceActivity implements
        SharedPreferences.OnSharedPreferenceChangeListener
{
    @Override
    protected void onCreate(Bundle icicle)
```

```
        {
            super.onCreate(icicle);
            getPreferenceManager().setSharedPreferencesName(LiveWallpaperService.PREFERENCES);
            addPreferencesFromResource(R.xml.settings);
            getPreferenceManager().getSharedPreferences().
                    registerOnSharedPreferenceChangeListener(this);
        }
        @Override
        protected void onDestroy()
        {
            getPreferenceManager().getSharedPreferences()
                    .unregisterOnSharedPreferenceChangeListener(this);
            super.onDestroy();
        }
        public void onSharedPreferenceChanged(SharedPreferences sharedPreferences, String key)
        {
        }
    }
```

本例还涉及到几个配置文件。首先应在 AndroidManifest.xml 文件中配置 LiveWallpaperService 和 LiveWallpaperSettings，代码如下：

```
    <service android:name="LiveWallpaperService" android:enabled="true"
        android:icon="@drawable/icon" android:label="@string/app_name"
        android:permission="android.permission.BIND_WALLPAPER">
        <intent-filter android:priority="1">
            <action android:name="android.service.wallpaper.WallpaperService" />
        </intent-filter>
        <meta-data android:name="android.service.wallpaper"
            android:resource="@xml/wallpaper" />
    </service>
    <activity android:label="@string/app_name" android:name=".LiveWallpaperSettings"
        android:theme="@android:style/Theme.Light.WallpaperSettings"
        android:exported="true" />
```

在 res\xml 目录中建立一个 settings.xml 文件，该文件用于设置 LiveWallpaperSettings 类的配置界面。settings.xml 文件的内容如下：

```
    <?xml version="1.0" encoding="utf-8"?>
    <PreferenceScreen
            xmlns:android="http://schemas.android.com/apk/res/android"
        android:title="@string/settings_title">
        <ListPreference
                android:key="preference_radius"
                android:title="@string/preference_radius_title"
                android:summary="@string/preference_radius_summary"
                android:entries="@array/radius_names"
                android:entryValues="@array/radius_values" />
    </PreferenceScreen>
```

最后还要在 res\xml 目录中建立一个 wallpaper.xml 文件，该文件需要在 AndroidManifest.xml 文件中的<meta-data>标签进行设置（就是 android:resource 属性的值）。wallpaper.xml 文件的内容如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<wallpaper xmlns:android="http://schemas.android.com/apk/res/android"
    android:thumbnail="@drawable/icon" android:description="@string/description"
    android:settingsActivity="net.blogjava.mobile.livewallpapers.LiveWallpaperSettings" />
```

19.4 小结

本章主要介绍了 Android 中比较有趣的两个功能：手势识别和 TTS。通过手势识别可以实现在屏幕上绘制简单的图形来输入复杂文本的功能，也可以利用手势来调用其他的应用程序。TTS 可以朗读指定的文本，但遗憾的是目前只支持英语等欧美语言。除此之外，还介绍了如何编写动态壁纸程序。