第 3 章　软件也要拼脸蛋，UI 开发的点点滴滴

我一直都认为程序员在软件的审美方面普遍都比较差，至少我个人就是如此。如果说要追究其根本原因，我觉得这是由于程序员的工作性质所导致的。每当我们看到一个软件时，不会像普通用户一样仅仅是关注一下它的界面以及有哪些功能。我们总是会不自觉地思考这些功能是如何实现的，很多在普通用户看来理所应当的功能，背后可能却需要非常复杂的算法来完成。以至于当别人唾骂一句，这软件做得真丑的时候，我们还可能赞叹一句，这功能做得好牛逼啊！

不过缺乏审美观毕竟不是一件值得炫耀的事情，在软件开发过程中，界面设计和功能开发同样重要。界面美观的应用程序不仅可以大大增加用户粘性，还能帮我们吸引到更多的新用户。而 Android 也是给我们提供了大量的 UI 开发工具，只要合理地使用它们，就可以编写出各种各样漂亮的界面。

在这里，我无法教会你如何提升自己的审美观，但我可以教会你怎样使用 Android 提供的 UI 开发工具来编写程序界面。想必你在上一章中反反复复地使用那几个按钮都快要吐了吧，本章我们就来学习更多的 UI 开发方面的知识。

3.1　该如何编写程序界面

Android 中有好几种编写程序界面的方式可供你选择。比如使用 DroidDraw，这是一种可视化的界面编辑工具，允许使用拖拽控件的方式来编写布局。Eclipse 和 Android Studio 中也有相应的可视化编辑器，和 DroidDraw 用法差不多，都是可以直接拖拽控件，并能在视图上修改控件属性的。不过以上的方式我都不推荐你使用，因为使用可视化编辑工具并不利于你去真正了解界面背后的实现原理，通常这种方式制作出的界面都不具有很好的屏幕适配性，而且当需要编写较为复杂的界面时，可视化编辑工具将很难胜任。因此本书中所有的界面我们都将使用最基本的方式去实现，即编写 XML 代码。等你完全掌握了使用 XML 来编写界面的方法之后，不管是进行高复杂度的界面实现，还是分析和修改当前现有界面，对你来说都将是手到擒来。听我这么说，你可能会觉得 Eclipse 中的可视化编辑器完全就是多余的嘛！其实也不是，你还是可以使用它来进行界面预览的，毕竟你无法直接通过 XML 就看

出界面的样子，而每修改一次界面就重新运行一遍程序显然又很耗时，这时你就可以好好地利用 Eclipse 的可视化编辑器了。

讲了这么多理论的东西，也是时候该学习一下到底如何编写程序界面了，我们就从 Android 中几种常见的控件开始吧。

3.2　常见控件的使用方法

Android 给我们提供了大量的 UI 控件，合理地使用这些控件就可以非常轻松地编写出相当不错的界面，下面我们就挑选几种常用的控件，详细介绍一下它们的使用方法。

首先新建一个 UIWidgetTest 项目，简单起见，我们还是允许 ADT 自动创建活动，活动名和布局名都使用默认值。别忘了将其他不相关的项目都关闭掉，始终养成这样一个良好的习惯。

3.2.1　TextView

TextView 可以说是 Android 中最简单的一个控件了，你在前面其实也已经和它打过了一些打交道。它主要用于在界面上显示一段文本信息，比如你在第一章看到的 Hello world！下面我们就来看一看关于 TextView 的更多用法。

将 activity_main.xml 中的代码改成如下所示：

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

    <TextView
        android:id="@+id/text_view"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="This is TextView" />

</LinearLayout>
```

外面的 LinearLayout 先忽略不看，在 TextView 中我们使用 android:id 给当前控件定义了一个唯一标识符，这个属性在上一章中已经讲解过了。然后使用 android:layout_width 指定了控件的宽度，使用 android:layout_height 指定了控件的高度。Android 中所有的控件都具有这两个属性，可选值有三种 match_parent、fill_parent 和 wrap_content，其中 match_parent 和 fill_parent 的意义相同，现在官方更加推荐使用 match_parent。match_parent 表示让当前控件

的大小和父布局的大小一样，也就是由父布局来决定当前控件的大小。wrap_content 表示让当前控件的大小能够刚好包含住里面的内容，也就是由控件内容决定当前控件的大小。所以上面的代码就表示让 TextView 的宽度和父布局一样宽，也就是手机屏幕的宽度，让 TextView 的高度足够包含住里面的内容就行。当然除了使用上述值，你也可以对控件的宽和高指定一个固定的大小，但是这样做有时会在不同手机屏幕的适配方面出现问题。接下来我们通过 android:text 指定了 TextView 中显示的文本内容，现在运行程序，效果如图 3.1 所示。

图 3.1

虽然指定的文本内容是正常显示了，不过我们好像没看出来 TextView 的宽度是和屏幕一样宽的。其实这是由于 TextView 中的文字默认是居左上角对齐的，虽然 TextView 的宽度充满了整个屏幕，可是从效果上完全看不出来。现在我们修改 TextView 的文字对齐方式，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" >

    <TextView
```

```
        android:id="@+id/text_view"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:gravity="center"
        android:text="This is TextView" />

</LinearLayout>
```

我们使用 android:gravity 来指定文字的对齐方式，可选值有 top、bottom、left、right、center 等，可以用“|”来同时指定多个值，这里我们指定的"center"，效果等同于"center_vertical|center_horizontal"，表示文字在垂直和水平方向都居中对齐。现在重新运行程序，效果如图 3.2 所示。

图　3.2

这也说明了，TextView 的宽度确实是和屏幕宽度一样的。

另外我们还可以对 TextView 中文字的大小和颜色进行修改，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" >

    <TextView
        android:id="@+id/text_view"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:gravity="center"
        android:textSize="24sp"
        android:textColor="#00ff00"
        android:text="This is TextView" />

</LinearLayout>
```

通过 android:textSize 属性可以指定文字的大小，通过 android:textColor 属性可以指定文字的颜色。重新运行程序，效果如图 3.3 所示。

图 3.3

当然 TextView 中还有很多其他的属性，这里我就不再一一介绍了，需要用到的时候去查阅文档就可以了。

3.2.2　Button

Button 是程序用于和用户进行交互的一个重要控件，相信你对这个控件已经是非常熟悉了，因为我们在上一章用了太多次 Button。它可配置的属性和 TextView 是差不多的，我们可以在 activity_main.xml 中这样加入 Button：

```
<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/button"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Button" />
</LinearLayout>
```

加入 Button 之后的界面如图 3.4 所示。

图　3.4

然后我们可以在 MainActivity 中为 Button 的点击事件注册一个监听器，如下所示：

```
public class MainActivity extends Activity {

    private Button button;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        button = (Button) findViewById(R.id.button);
        button.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
                // 在此处添加逻辑
            }
        });
    }

}
```

这样每当点击按钮时，就会执行监听器中的 onClick()方法，我们只需要在这个方法中加入待处理的逻辑就行了。如果你不喜欢使用匿名类的方式来注册监听器，也可以使用实现接口的方式来进行注册，代码如下所示：

```
public class MainActivity extends Activity implements OnClickListener {

    private Button button;

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

    @Override
    public void onClick(View v) {
        switch (v.getId()) {
        case R.id.button:
```

```
            // 在此处添加逻辑
            break;
        default:
            break;
        }
    }

}
```

这两种写法都可以实现对按钮点击事件的监听，至于使用哪一种就全凭你喜好了。

3.2.3　EditText

EditText 是程序用于和用户进行交互的另一个重要控件，它允许用户在控件里输入和编辑内容，并可以在程序中对这些内容进行处理。EditText 的应用场景应该算是非常普遍了，发短信、发微博、聊 QQ 等等，在进行这些操作时，你不得不使用到 EditText。那我们来看一看如何在界面上加入 EditText 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" >

    ......

    <EditText
        android:id="@+id/edit_text"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        />
</LinearLayout>
```

其实看到这里，我估计你已经总结出 Android 控件的使用规律了，基本上用法都很相似，给控件定义一个 id，再指定下控件的宽度和高度，然后再适当加入些控件特有的属性就差不多了。所以使用 XML 来编写界面其实一点都不难，完全可以不用借助任何可视化工具来实现。现在重新运行一下程序，EditText 就已经在界面上显示出来了，并且我们是可以在里面输入内容的，如图 3.5 所示。

图 3.5

细心的你平时应该会留意到，一些做得比较人性化的软件会在输入框里显示一些提示性的文字，然后一旦用户输入了任何内容，这些提示性的文字就会消失。这种提示功能在 Android 里是非常容易实现的，我们甚至不需要做任何的逻辑控制，因为系统已经帮我们都处理好了。修改 activity_main.xml，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">

        ......

    <EditText
        android:id="@+id/edit_text"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="Type something here"
        />
</LinearLayout>
```

这里使用 android:hint 属性来指定了一段提示性的文本，然后重新运行程序，效果如图 3.6 所示。

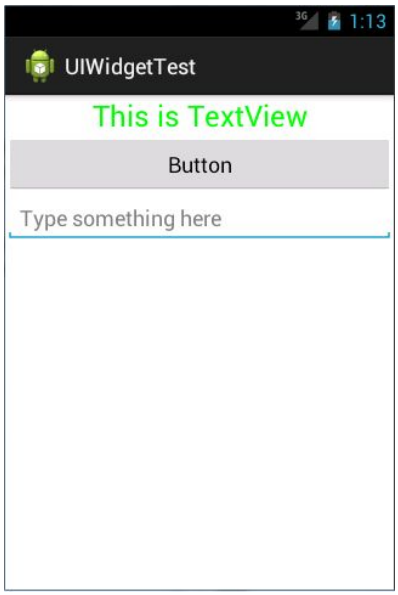

图　3.6

可以看到，EditText 中显示了一段提示性文本，然后当我们输入任何内容时，这段文本就会自动消失。

不过随着输入的内容不断增多，EditText 会被不断地拉长。这时由于 EditText 的高度指定的是 wrap_content，因此它总能包含住里面的内容，但是当输入的内容过多时，界面就会变得非常难看。我们可以使用 android:maxLines 属性来解决这个问题，修改 activity_main.xml，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" >

    ......

    <EditText
        android:id="@+id/edit_text"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
```

```
        android:hint="Type something here"
        android:maxLines="2"
        />
</LinearLayout>
```

这里通过 android:maxLines 指定了 EditText 的最大行数为两行，这样当输入的内容超过两行时，文本就会向上滚动，而 EditText 则不会再继续拉伸，如图 3.7 所示。

图 3.7

我们还可以结合使用 EditText 与 Button 来完成一些功能，比如通过点击按钮来获取 EditText 中输入的内容。修改 MainActivity 中的代码，如下所示：

```
public class MainActivity extends Activity implements OnClickListener {

    private Button button;

    private EditText editText;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        button = (Button) findViewById(R.id.button);
```

```
        editText = (EditText) findViewById(R.id.edit_text);
        button.setOnClickListener(this);
    }

    @Override
    public void onClick(View v) {
        switch (v.getId()) {
        case R.id.button:
            String inputText = editText.getText().toString();
            Toast.makeText(MainActivity.this, inputText,
Toast.LENGTH_SHORT).show();
            break;
        default:
            break;
        }
    }

}
```

首先通过 findViewById()方法得到 EditText 的实例，然后在按钮的点击事件里调用 EditText 的 getText()方法获取到输入的内容，再调用 toString()方法转换成字符串，最后仍然还是老方法，使用 Toast 将输入的内容显示出来。

重新运行程序，在 EditText 中输入一段内容，然后点击按钮，效果如图 3.8 所示。

图　3.8

3.2.4 ImageView

ImageView 是用于在界面上展示图片的一个控件，通过它可以让我们的程序界面变得更加丰富多彩。学习这个控件需要提前准备好一些图片，由于目前 drawable 文件夹下已经有一张 ic_launcher.png 图片了，那我们就先在界面上展示这张图吧，修改 activity_main.xml，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" >

    ......

    <ImageView
        android:id="@+id/image_view"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:src="@drawable/ic_launcher"
        />
</LinearLayout>
```

可以看到，这里使用 android:src 属性给 ImageView 指定了一张图片，并且由于图片的宽和高都是未知的，所以将 ImageView 的宽和高都设定为 wrap_content，这样保证了不管图片的尺寸是多少都可以完整地展示出来。重新运行程序，效果如图 3.9 所示。

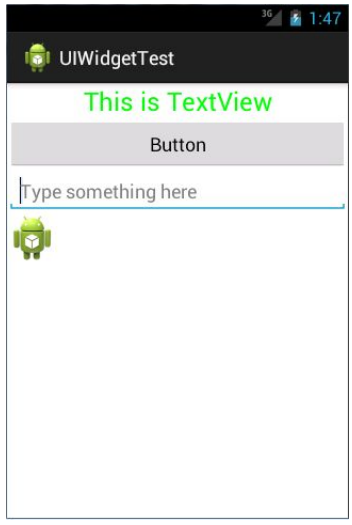

图 3.9

我们还可以在程序中通过代码动态地更改 ImageView 中的图片。这里我准备了另外一张图片，jelly_bean.png，将它复制到 res/drawable-hdpi 目录下，然后修改 MainActivity 的代码，如下所示：

```
public class MainActivity extends Activity implements OnClickListener {

    private Button button;

    private EditText editText;

    private ImageView imageView;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        button = (Button) findViewById(R.id.button);
        editText = (EditText) findViewById(R.id.edit_text);
        imageView  = (ImageView) findViewById(R.id.image_view);
        button.setOnClickListener(this);
    }

    @Override
    public void onClick(View v) {
        switch (v.getId()) {
        case R.id.button:
            imageView.setImageResource(R.drawable.jelly_bean);
            break;
        default:
            break;
        }
    }

}
```

在按钮的点击事件里，通过调用 ImageView 的 setImageResource()方法将显示的图片改成 jelly_bean，现在重新运行程序，然后点击一下按钮，就可以看到 ImageView 中显示的图片改变了，如图 3.10 所示。

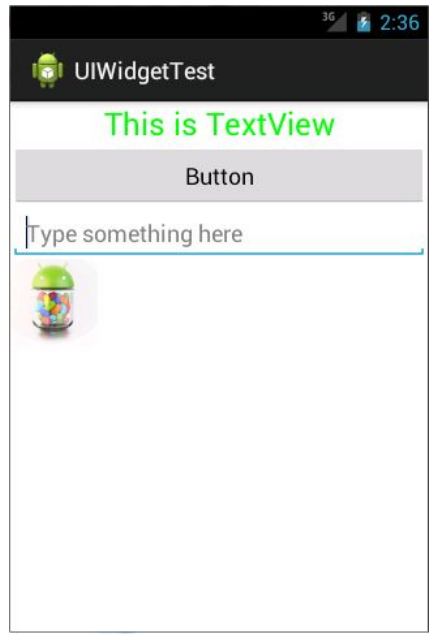

图 3.10

3.2.5 ProgressBar

ProgressBar 用于在界面上显示一个进度条，表示我们的程序正在加载一些数据。它的用法也非常简单，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" >

    ......

    <ProgressBar
        android:id="@+id/progress_bar"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        />
</LinearLayout>
```

重新运行程序，会看到屏幕中有一个圆形进度条正在旋转，如图 3.11 所示。

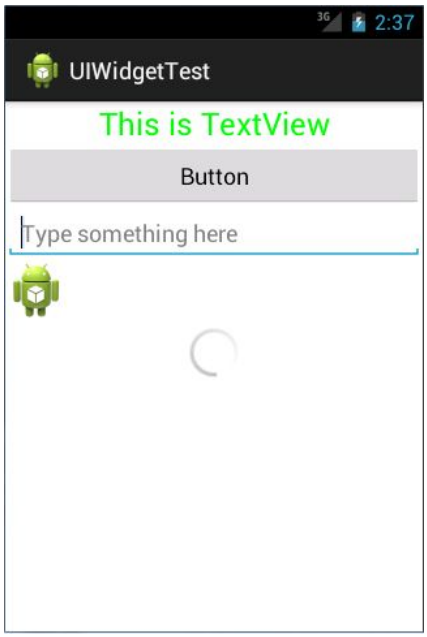

图　3.11

这时你可能会问，旋转的进度条表明我们的程序正在加载数据，那数据总会有加载完的时候吧，如何才能让进度条在数据加载完成时消失呢？这里我们就需要用到一个新的知识点，Android 控件的可见属性。所有的 Android 控件都具有这个属性，可以通过 android:visibility 进行指定，可选值有三种，visible、invisible 和 gone。visible 表示控件是可见的，这个值是默认值，不指定 android:visibility 时，控件都是可见的。invisible 表示控件不可见，但是它仍然占据着原来的位置和大小，可以理解成控件变成透明状态了。gone 则表示控件不仅不可见，而且不再占用任何屏幕空间。我们还可以通过代码来设置控件的可见性，使用的是 setVisibility()方法，可以传入 View.VISIBLE、View.INVISIBLE 和 View.GONE 三种值。

接下来我们就来尝试实现，点击一下按钮让进度条消失，再点击一下按钮让进度条出现的这种效果。修改 MainActivity 中的代码，如下所示：

```
public class MainActivity extends Activity implements OnClickListener {

    private Button button;

    private EditText editText;

    private ImageView imageView;
```

```
    private ProgressBar progressBar;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        button = (Button) findViewById(R.id.button);
        editText = (EditText) findViewById(R.id.edit_text);
        imageView  = (ImageView) findViewById(R.id.image_view);
        progressBar = (ProgressBar) findViewById(R.id.progress_bar);
        button.setOnClickListener(this);
    }

    @Override
    public void onClick(View v) {
        switch (v.getId()) {
        case R.id.button:
            if (progressBar.getVisibility() == View.GONE) {
                progressBar.setVisibility(View.VISIBLE);
            } else {
                progressBar.setVisibility(View.GONE);
            }
            break;
        default:
            break;
        }
    }

}
```

在按钮的点击事件中，我们通过 getVisibility()方法来判断 ProgressBar 是否可见，如果可见就将 ProgressBar 隐藏掉，如果不可见就将 ProgressBar 显示出来。重新运行程序，然后不断地点击按钮，你就会看到进度条会在显示与隐藏之间来回切换。

另外，我们还可以给 ProgressBar 指定不同的样式，刚刚是圆形进度条，通过 style 属性可以将它指定成水平进度条，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" >
```

```
    ……

    <ProgressBar
        android:id="@+id/progress_bar"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        style="?android:attr/progressBarStyleHorizontal"
        android:max="100"
        />
</LinearLayout>
```

指定成水平进度条后，我们还可以通过 android:max 属性给进度条设置一个最大值，然后在代码中动态地更改进度条的进度。修改 MainActivity 中的代码，如下所示：

```
public class MainActivity extends Activity implements OnClickListener {
    ……
    @Override
    public void onClick(View v) {
        switch (v.getId()) {
        case R.id.button:
            int progress = progressBar.getProgress();
            progress = progress + 10;
            progressBar.setProgress(progress);
            break;
        default:
            break;
        }
    }
}
```

每点击一次按钮，我们就获取进度条的当前进度，然后在现有的进度上加 10 作为更新后的进度。重新运行程序，点击数次按钮后，效果如图 3.12 所示。

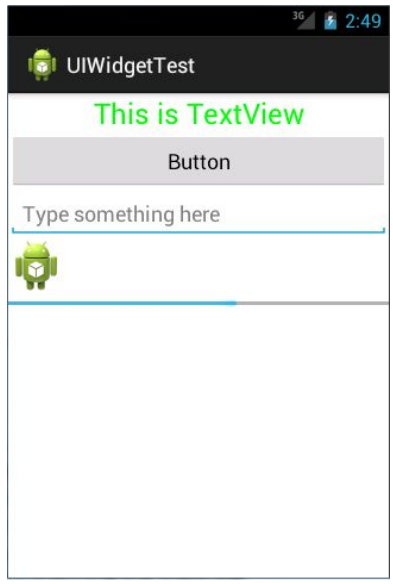

图 3.12

ProgressBar 还有几种其他的样式，你可以自己去尝试一下。

3.2.6 AlertDialog

AlertDialog 可以在当前的界面弹出一个对话框，这个对话框是置顶于所有界面元素之上的，能够屏蔽掉其他控件的交互能力，因此一般 AlertDialog 都是用于提示一些非常重要的内容或者警告信息。比如为了防止用户误删重要内容，在删除前弹出一个确认对话框。下面我们来学习一下它的用法，修改 MainActivity 中的代码，如下所示：

```
public class MainActivity extends Activity implements OnClickListener {
    ……
    @Override
    public void onClick(View v) {
        switch (v.getId()) {
        case R.id.button:
            AlertDialog.Builder dialog = new AlertDialog.Builder
(MainActivity.this);
            dialog.setTitle("This is Dialog");
            dialog.setMessage("Something important.");
            dialog.setCancelable(false);
            dialog.setPositiveButton("OK", new DialogInterface.
OnClickListener() {
```

```
                @Override
                public void onClick(DialogInterface dialog, int which) {
                }
            });
            dialog.setNegativeButton("Cancel", new DialogInterface.
OnClickListener() {
                @Override
                public void onClick(DialogInterface dialog, int which) {
                }
            });
            dialog.show();
            break;
        default:
            break;
        }
    }
}
```

首先通过 AlertDialog.Builder 创建出一个 AlertDialog 的实例，然后可以为这个对话框设置标题、内容、可否取消等属性，接下来调用 setPositiveButton()方法为对话框设置确定按钮的点击事件，调用 setNegativeButton()方法设置取消按钮的点击事件，最后调用 show()方法将对话框显示出来。重新运行程序，点击按钮后，效果如图 3.13 所示。

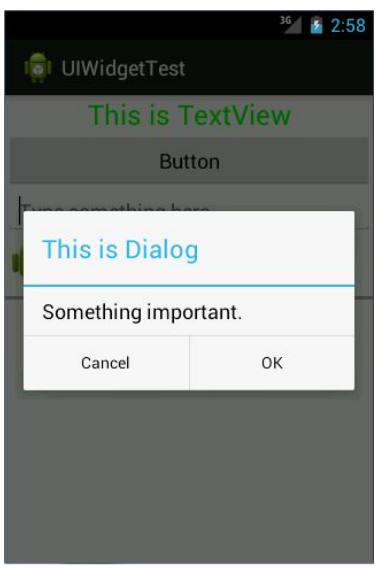

图　3.13

3.2.7 ProgressDialog

ProgressDialog 和 AlertDialog 有点类似，都可以在界面上弹出一个对话框，都能够屏蔽掉其他控件的交互能力。不同的是，ProgressDialog 会在对话框中显示一个进度条，一般是用于表示当前操作比较耗时，让用户耐心地等待。它的用法和 AlertDialog 也比较相似，修改 MainActivity 中的代码，如下所示：

```
public class MainActivity extends Activity implements OnClickListener {
    ……
    @Override
    public void onClick(View v) {
        switch (v.getId()) {
        case R.id.button:
            ProgressDialog progressDialog = new ProgressDialog
(MainActivity.this);
            progressDialog.setTitle("This is ProgressDialog");
            progressDialog.setMessage("Loading...");
            progressDialog.setCancelable(true);
            progressDialog.show();
            break;
        default:
            break;
        }
    }
}
```

可以看到，这里也是先构建出一个 ProgressDialog 对象，然后同样可以设置标题、内容、可否取消等属性，最后也是通过调用 show()方法将 ProgressDialog 显示出来。重新运行程序，点击按钮后，效果如图 3.14 所示。

注意如果在 setCancelable()中传入了 false，表示 ProgressDialog 是不能通过 Back 键取消掉的，这时你就一定要在代码中做好控制，当数据加载完成后必须要调用 ProgressDialog 的 dismiss()方法来关闭对话框，否则 ProgressDialog 将会一直存在。

图　3.14

好了，关于 Android 控件的使用，我要讲的就只有这么多。一节内容就想覆盖 Android 控件所有的相关知识不太现实，同样一口气就想学会所有 Android 控件的使用方法也不太现实。本节所讲的内容对于你来说只是起到了一个引导的作用，你还需要在以后的学习和工作中不断地摸索，通过查阅文档以及网上搜索的方式学习更多控件的更多用法。当然，当本书后面有涉及到一些我们前面没学过的控件和相关用法时，我仍然会在相应的章节做详细的讲解。

经验值：+1000　升级！（由小菜鸟升级至菜鸟）　目前经验值：3905

级别：菜鸟

赢得宝物：战胜常见控件矿守护者。拾取常见控件矿守护者掉落的宝物，一套尚未拼完的智力拼图玩具、一本书《设计师心理学——神人界面篇》，以及一颗大型信心增强大力丸。常见控件矿守护者私下里把自己称作重要控件矿守护者，但见多识广的神界精灵们指出这些“重要控件矿”事实上很常见，而且取之不尽用之不竭，根本不需要派专人守护。而常见控件矿守护者则辩称正是由于他们世世代代兢兢业业的守护，并且建立了大量的神界地质保护区，才让这些重要控件矿得以繁衍生息，变得常见。智力拼图玩具很快就拼好了。《设计师心理学》也写得通俗易懂，路上可以慢慢看。继续前进。

3.3 详解四种基本布局

一个丰富的界面总是要由很多个控件组成的，那我们如何才能让各个控件都有条不紊地摆放在界面上，而不是乱糟糟的呢？这就需要借助布局来实现了。布局是一种可用于放置很多控件的容器，它可以按照一定的规律调整内部控件的位置，从而编写出精美的界面。当然，布局的内部除了放置控件外，也可以放置布局，通过多层布局的嵌套，我们就能够完成一些比较复杂的界面实现，示意图 3.15 很好地展示了它们之间的关系。

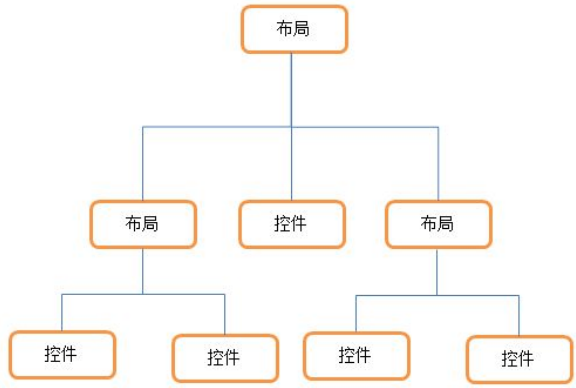

图 3.15

下面我们来详细讲解下 Android 中四种最基本的布局。先做好准备工作，新建一个 UILayoutTest 项目，并让 ADT 自动帮我们创建好活动，活动名和布局名都使用默认值。

3.3.1 LinearLayout

LinearLayout 又称作线性布局，是一种非常常用的布局。正如它名字所描述的一样，这个布局会将它所包含的控件在线性方向上依次排列。相信你之前也已经注意到了，我们在上一节中学习控件用法时，所有的控件就都是放在 LinearLayout 布局里的，因此上一节中的控件也确实是在垂直方向上线性排列的。

既然是线性排列，肯定就不仅只有一个方向，那为什么上一节中的控件都是在垂直方向排列的呢？这是由于我们通过 android:orientation 属性指定了排列方向是 vertical，如果指定的是 horizontal，控件就会在水平方向上排列了。下面我们通过实战来体会一下，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/button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Button 1" />

    <Button
        android:id="@+id/button2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Button 2" />

    <Button
        android:id="@+id/button3"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Button 3" />

</LinearLayout>
```

我们在 LinearLayout 中添加了三个 Button，每个 Button 的长和宽都是 wrap_content，并指定了排列方向是 vertical。现在运行一下程序，效果如图 3.16 所示。

图　3.16

然后我们修改一下 LinearLayout 的排列方向，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="horizontal" >

    ......

</LinearLayout>
```

将 android:orientation 属性的值改成了 horizontal，这就意味着要让 LinearLayout 中的控件在水平方向上依次排列，当然如果不指定 android:orientation 属性的值，默认的排列方向就是 horizontal。重新运行一下程序，效果如图 3.17 所示。

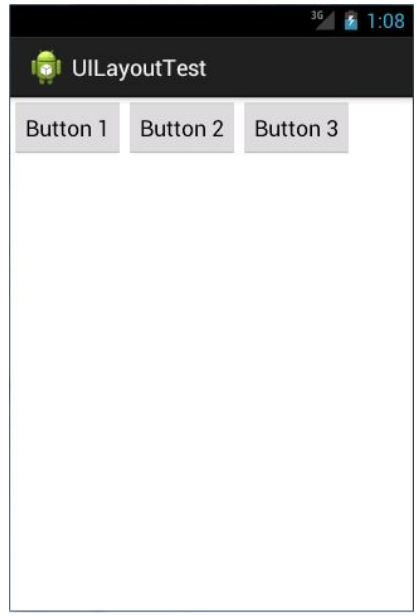

图 3.17

这里需要注意，如果 LinearLayout 的排列方向是 horizontal，内部的控件就绝对不能将宽度指定为 match_parent，因为这样的话单独一个控件就会将整个水平方向占满，其他的控件就没有可放置的位置了。同样的道理，如果 LinearLayout 的排列方向是 vertical，内部的控件就不能将高度指定为 match_parent。

了解了 LinearLayout 的排列规律，我们再来学习一下它的几个关键属性的用法吧。

首先来看 android:layout_gravity 属性，它和我们上一节中学到的 android:gravity 属性看

起来有些相似，这两个属性有什么区别呢？其实从名字上就可以看出，android:gravity 是用于指定文字在控件中的对齐方式，而 android:layout_gravity 是用于指定控件在布局中的对齐方式。android:layout_gravity 的可选值和 android:gravity 差不多，但是需要注意，当 LinearLayout 的排列方向是 horizontal 时，只有垂直方向上的对齐方式才会生效，因为此时水平方向上的长度是不固定的，每添加一个控件，水平方向上的长度都会改变，因而无法指定该方向上的对齐方式。同样的道理，当 LinearLayout 的排列方向是 vertical 时，只有水平方向上的对齐方式才会生效。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="horizontal" >

    <Button
        android:id="@+id/button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="top"
        android:text="Button 1" />

    <Button
        android:id="@+id/button2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center_vertical"
        android:text="Button 2" />

    <Button
        android:id="@+id/button3"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="bottom"
        android:text="Button 3" />

</LinearLayout>
```

由于目前 LinearLayout 的排列方向是 horizontal，因此我们只能指定垂直方向上的排列方向，将第一个 Button 的对齐方式指定为 top，第二个 Button 的对齐方式指定为 center_vertical，第三个 Button 的对齐方式指定为 bottom。重新运行程序，效果如图 3.18 所示。

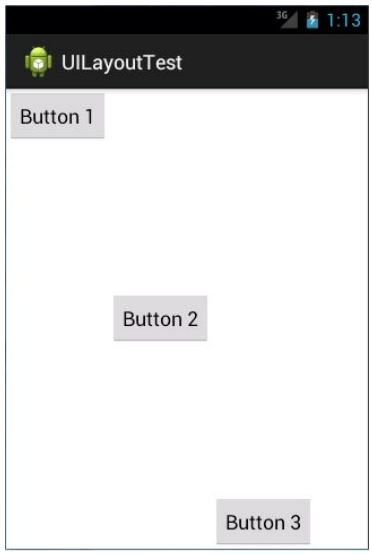

图 3.18

接下来我们学习下 LinearLayout 中的另一个重要属性，android:layout_weight。这个属性允许我们使用比例的方式来指定控件的大小，它在手机屏幕的适配性方面可以起到非常重要的作用。比如我们正在编写一个消息发送界面，需要一个文本编辑框和一个发送按钮，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="horizontal" >

    <EditText
        android:id="@+id/input_message"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:hint="Type something"
        />

    <Button
        android:id="@+id/send"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
```

```
        android:layout_weight="1"
        android:text="Send"
        />

</LinearLayout>
```

你会发现，这里竟然将 EditText 和 Button 的宽度都指定成了 0，这样文本编辑框和按钮还能显示出来吗？不用担心，由于我们使用了 android:layout_weight 属性，此时控件的宽度就不应该再由 android:layout_width 来决定，这里指定成 0 是一种比较规范的写法。

然后我们在 EditText 和 Button 里都将 android:layout_weight 属性的值指定为 1，这表示 EditText 和 Button 将在水平方向平分宽度，重新运行下程序，你会看到如图 3.19 所示的效果。

图　3.19

为什么将 android:layout_weight 属性的值同时指定为 1 就会平分屏幕宽度呢？其实原理也很简单，系统会先把 LinearLayout 下所有控件指定的 layout_weight 值相加，得到一个总值，然后每个控件所占大小的比例就是用该控件的 layout_weight 值除以刚才算出的总值。因此如果想让 EditText 占据屏幕宽度的 3/5，Button 占据屏幕宽度的 2/5，只需要将 EditText 的 layout_weight 改成 3，Button 的 layout_weight 改成 2 就可以了。

我们还可以通过指定部分控件的 layout_weight 值，来实现更好的效果。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="horizontal" >

    <EditText
        android:id="@+id/input_message"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:hint="Type something"
        />

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

</LinearLayout>
```

这里我们仅指定了 EditText 的 android:layout_weight 属性，并将 Button 的宽度改回 wrap_content。这表示 Button 的宽度仍然按照 wrap_content 来计算，而 EditText 则会占满屏幕所有的剩余空间。使用这种方式编写的界面，不仅在各种屏幕的适配方面会非常好，而且看起来也更加舒服，重新运行程序，效果如图 3.20 所示。

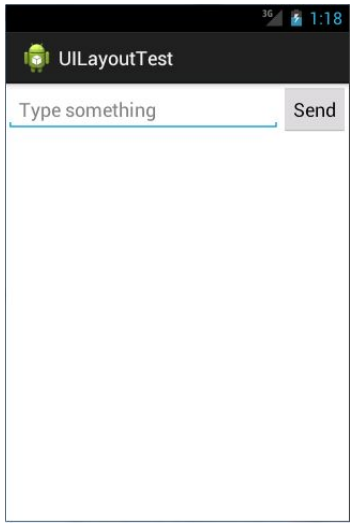

图 3.20

3.3.2　RelativeLayout

RelativeLayout 又称作相对布局，也是一种非常常用的布局。和 LinearLayout 的排列规则不同，RelativeLayout 显得更加随意一些，它可以通过相对定位的方式让控件出现在布局的任何位置。也正因为如此，RelativeLayout 中的属性非常多，不过这些属性都是有规律可循的，其实并不难理解和记忆。我们还是通过实践来体会一下，修改 activity_main.xml 中的代码，如下所示：

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

    <Button
        android:id="@+id/button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_alignParentTop="true"
        android:text="Button 1" />

    <Button
        android:id="@+id/button2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentRight="true"
        android:layout_alignParentTop="true"
        android:text="Button 2" />

    <Button
        android:id="@+id/button3"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerInParent="true"
        android:text="Button 3" />

    <Button
        android:id="@+id/button4"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentBottom="true"
```

```
        android:layout_alignParentLeft="true"
        android:text="Button 4" />

    <Button
        android:id="@+id/button5"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentBottom="true"
        android:layout_alignParentRight="true"
        android:text="Button 5" />

</RelativeLayout>
```

我想以上代码已经不需要我再做过多解释了，因为实在是太好理解了，我们让 Button 1 和父布局的左上角对齐，Button 2 和父布局的右上角对齐，Button 3 居中显示，Button 4 和父布局的左下角对齐，Button 5 和父布局的右下角对齐。虽然 android:layout_alignParentLeft、android:layout_alignParentTop、android:layout_alignParentRight、android:layout_alignParentBottom、android:layout_centerInParent 这几个属性我们之前都没接触过，可是它们的名字已经完全说明了它们的作用。重新运行程序，效果如图 3.21 所示。

图 3.21

上面例子中的每个控件都是相对于父布局进行定位的，那控件可不可以相对于控件进行定位呢？当然是可以的，修改 activity_main.xml 中的代码，如下所示：

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

    <Button
        android:id="@+id/button3"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerInParent="true"
        android:text="Button 3" />

    <Button
        android:id="@+id/button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_above="@id/button3"
        android:layout_toLeftOf="@id/button3"
        android:text="Button 1" />

    <Button
        android:id="@+id/button2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_above="@id/button3"
        android:layout_toRightOf="@id/button3"
        android:text="Button 2" />

    <Button
        android:id="@+id/button4"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@id/button3"
        android:layout_toLeftOf="@id/button3"
        android:text="Button 4" />

    <Button
        android:id="@+id/button5"
```

```
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@id/button3"
        android:layout_toRightOf="@id/button3"
        android:text="Button 5" />

</RelativeLayout>
```

这次的代码稍微复杂一点，不过仍然是有规律可循的。android:layout_above 属性可以让一个控件位于另一个控件的上方，需要为这个属性指定相对控件 id 的引用，这里我们填入了@id/button3，表示让该控件位于 Button 3 的上方。其他的属性也都是相似的，android:layout_below 表示让一个控件位于另一个控件的下方，android:layout_toLeftOf 表示让一个控件位于另一个控件的左侧，android:layout_toRightOf 表示让一个控件位于另一个控件的右侧。注意，当一个控件去引用另一个控件的 id 时，该控件一定要定义在引用控件的后面，不然会出现找不到 id 的情况。重新运行程序，效果如图 3.22 所示。

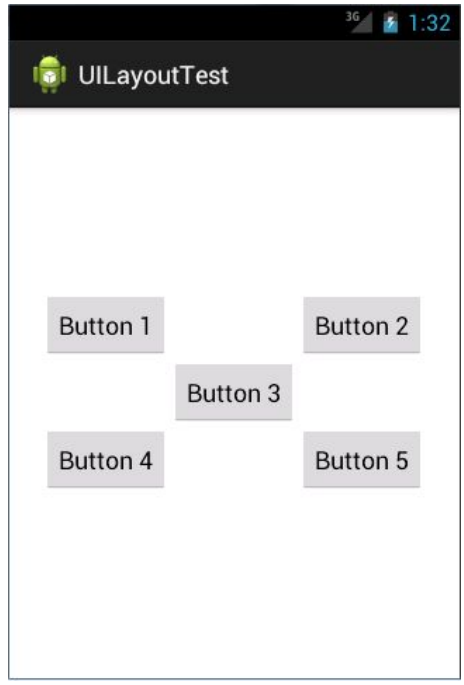

图 3.22

RelativeLayout 中还有另外一组相对于控件进行定位的属性，android:layout_alignLeft 表示让一个控件的左边缘和另一个控件的左边缘对齐，android:layout_alignRight 表示让一个控

件的右边缘和另一个控件的右边缘对齐，还有 android:layout_alignTop 和 android:layout_alignBottom，道理都是一样的，我就不再多说，这几个属性就留给你自己去尝试一下了。

好了，正如我前面所说，RelativeLayout 中的属性虽然多，但都是有规律可循的，所以学起来一点都不觉得吃力吧？

3.3.3　FrameLayout

FrameLayout 相比于前面两种布局就简单太多了，因此它的应用场景也少了很多。这种布局没有任何的定位方式，所有的控件都会摆放在布局的左上角。让我们通过例子来看一看吧，修改 activity_main.xml 中的代码，如下所示：

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

    <Button
        android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Button"
        />

    <ImageView
        android:id="@+id/image_view"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:src="@drawable/ic_launcher"
        />

</FrameLayout>
```

FrameLayout 中只是放置了一个按钮和一张图片，重新运行程序，效果如图 3.23 所示。

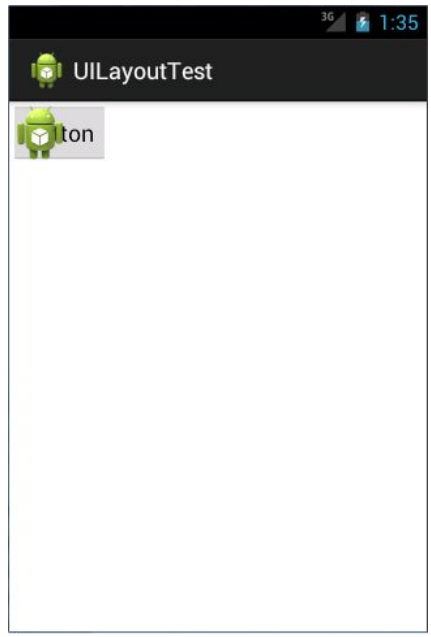

图 3.23

可以看到，按钮和图片都是位于布局的左上角。由于图片是在按钮之后添加的，因此图片压在了按钮的上面。

你可能会觉得，这个布局能有什么作用呢？确实，它的应用场景并不多，不过在下一章中介绍碎片的时候，我们还是可以用到它的。

3.3.4 TableLayout

TableLayout 允许我们使用表格的方式来排列控件，这种布局也不是很常用，你只需要了解一下它的基本用法就可以了。既然是表格，那就一定会有行和列，在设计表格时我们尽量应该让每一行都拥有相同的列数，这样的表格也是最简单的。不过有时候事情并非总会顺从我们的心意，当表格的某行一定要有不相等的列数时，就需要通过合并单元格的方式来应对。

比如我们正在设计一个登录界面，允许用户输入账号密码后登录，就可以将 activity_main.xml 中的代码改成如下所示：

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

```
    <TableRow>

        <TextView
            android:layout_height="wrap_content"
            android:text="Account:" />

        <EditText
            android:id="@+id/account"
            android:layout_height="wrap_content"
            android:hint="Input your account" />
    </TableRow>

    <TableRow>

        <TextView
            android:layout_height="wrap_content"
            android:text="Password:" />

        <EditText
            android:id="@+id/password"
            android:layout_height="wrap_content"
            android:inputType="textPassword" />
    </TableRow>

    <TableRow>

        <Button
            android:id="@+id/login"
            android:layout_height="wrap_content"
            android:layout_span="2"
            android:text="Login" />
    </TableRow>

</TableLayout>
```

在 TableLayout 中每加入一个 TableRow 就表示在表格中添加了一行，然后在 TableRow 中每加入一个控件，就表示在该行中加入了一列，TableRow 中的控件是不能指定宽度的。这里我们将表格设计成了三行两列的格式，第一行有一个 TextView 和一个用于输入账号的 EditText，第二行也有一个 TextView 和一个用于输入密码的 EditText，我们通过将 android:inputType 属性的值指定为 textPassword，把 EditText 变为密码输入框。可是第三行只

有一个用于登录的按钮，前两行都有两列，第三行只有一列，这样的表格就会很难看，而且结构也非常不合理。这时就需要通过对单元格进行合并来解决这个问题，使用android:layout_span="2"让登录按钮占据两列的空间，就可以保证表格结构的合理性了。重新运行程序，效果如图 3.24 所示。

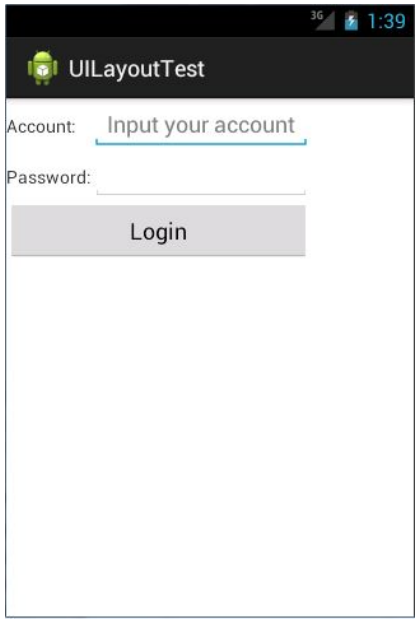

图 3.24

不过从图中可以看出，当前的登录界面并没有充分利用屏幕的宽度，右侧还空出了一块区域，这也难怪，因为在 TableRow 中我们无法指定控件的宽度。这时使用android:stretchColumns 属性就可以很好地解决这个问题，它允许将 TableLayout 中的某一列进行拉伸，以达到自动适应屏幕宽度的作用。修改 activity_main.xml 中的代码，如下所示：

```
<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:stretchColumns="1"
    >

    ......

</TableLayout>
```

这里将 android:stretchColumns 的值指定为 1，表示如果表格不能完全占满屏幕宽度，就

将第二列进行拉伸。没错！指定成 1 就是拉伸第二列，指定成 0 就是拉伸第一列，不要以为这里我写错了哦。重新运行程序，效果如图 3.25 所示。

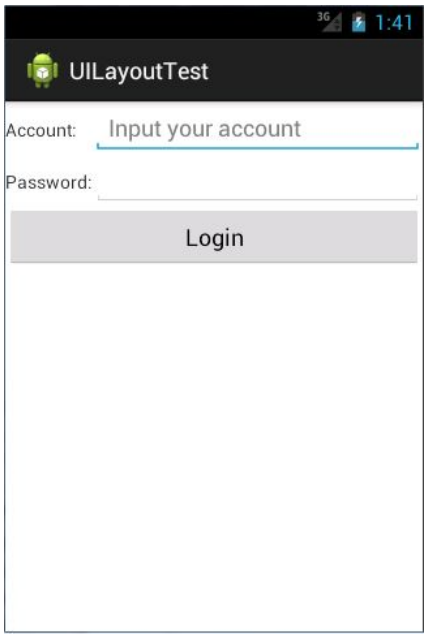

图　3.25

好了，关于布局我也就只准备讲这么多了，Android 中其实还有一个 AbsoluteLayout，不过这个布局官方已经不推荐使用了，因此我们直接将它忽略就好。

经验值：+900　　目前经验值：4705

级别：菜鸟

赢得宝物：战胜布局法师。拾取布局法师掉落的宝物，一顶羽毛法师帽，一杆鹿骨法师杖，以及 25 张神界大型快餐连锁店“神当闲”的优惠券。戴上羽毛法师帽可以让你的脑子里冒出大量毫无价值甚至臭不可闻的低阶灵感，然后配合使用鹿骨法师杖可以让你把每 100 个低阶灵感合并成一个无异味的中级灵感，以应付日常生活。布局法师在神界属于低阶法师，但他尊老爱幼，勇于扶老人家过马路，在神界拥有很好的人缘。

3.4　系统控件不够用？创建自定义控件

在前面两节我们已经学习了 Android 中的一些常见控件以及基本布局的用法，不过当时我们并没有关注这些控件和布局的继承结构，现在是时候应该看一下了，如图 3.26 所示。

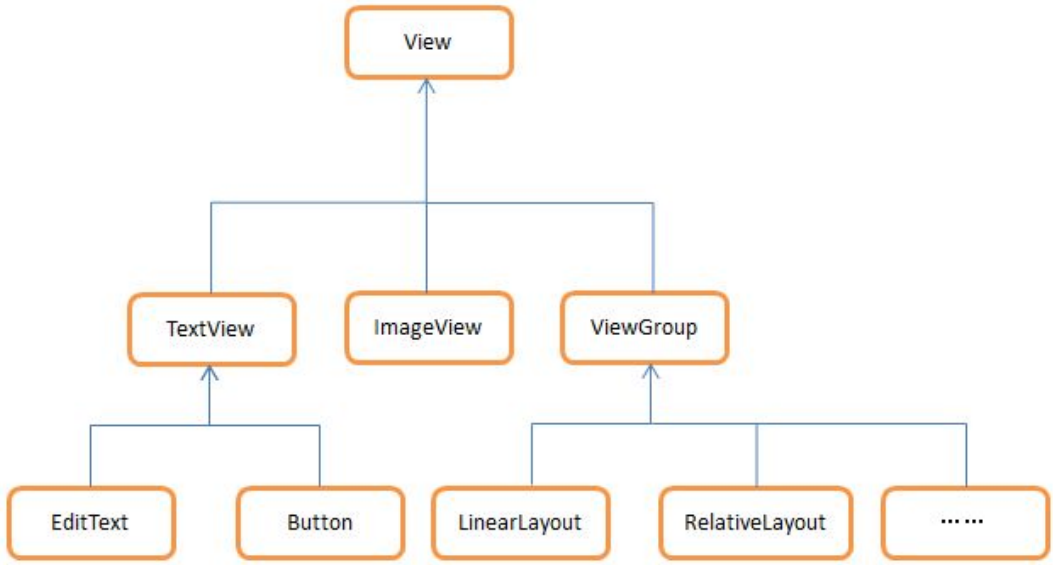

图 3.26

可以看到，我们所用的所有控件都是直接或间接继承自 View 的，所用的所有布局都是直接或间接继承自 ViewGroup 的。View 是 Android 中一种最基本的 UI 组件，它可以在屏幕上绘制一块矩形区域，并能响应这块区域的各种事件，因此，我们使用的各种控件其实就是在 View 的基础之上又添加了各自特有的功能。而 ViewGroup 则是一种特殊的 View，它可以包含很多的子 View 和子 ViewGroup，是一个用于放置控件和布局的容器。

这个时候我们就可以思考一下，如果系统自带的控件并不能满足我们的需求时，可不可以利用上面的继承结构来创建自定义控件呢？答案是肯定的，下面我们就来学习一下创建自定义控件的两种简单方法。先将准备工作做好，创建一个 UICustomViews 项目。

3.4.1 引入布局

如果你用过 iPhone 应该会知道，几乎每一个 iPhone 应用的界面顶部都会有一个标题栏，标题栏上会有一到两个按钮可用于返回或其他操作（iPhone 没有实体返回键）。现在很多的 Android 程序也都喜欢模仿 iPhone 的风格，在界面的顶部放置一个标题栏。虽然 Android 系统已经给每个活动提供了标题栏功能，但这里我们仍然决定不使用它，而是创建一个自定义的标题栏。

经过前面两节的学习，我想创建一个标题栏布局对你来说已经不是什么困难的事情了，只需要加入两个 Button 和一个 TextView，然后在布局中摆放好就可以了。可是这样做却存在着一个问题，一般我们的程序中可能有很多个活动都需要这样的标题栏，如果在每个活动

的布局中都编写一遍同样的标题栏代码，明显就会导致代码的大量重复。这个时候我们就可以使用引入布局的方式来解决这个问题，新建一个布局 title.xml，代码如下所示：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:background="@drawable/title_bg" >

    <Button
        android:id="@+id/title_back"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:layout_margin="5dip"
        android:background="@drawable/back_bg"
        android:text="Back"
        android:textColor="#fff" />

    <TextView
        android:id="@+id/title_text"
        android:layout_width="0dip"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:layout_weight="1"
        android:gravity="center"
        android:text="Title Text"
        android:textColor="#fff"
        android:textSize="24sp" />

    <Button
        android:id="@+id/title_edit"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:layout_margin="5dip"
        android:background="@drawable/edit_bg"
        android:text="Edit"
        android:textColor="#fff" />

</LinearLayout>
```

可以看到，我们在 LinearLayout 中分别加入了两个 Button 和一个 TextView，左边的 Button 可用于返回，右边的 Button 可用于编辑，中间的 TextView 则可以显示一段标题文本。上面的代码中大多数的属性你都已经是见过的，下面我来说明一下几个之前没有讲过的属性。android:background 用于为布局或控件指定一个背景，可以使用颜色或图片来进行填充，这里我提前准备好了三张图片，title_bg.png、back_bg.png 和 edit_bg.png，分别用于作为标题栏、返回按钮和编辑按钮的背景。另外在两个 Button 中我们都使用了 android:layout_margin 这个属性，它可以指定控件在上下左右方向上偏移的距离，当然也可以使用 android:layout_marginLeft 或 android:layout_marginTop 等属性来单独指定控件在某个方向上偏移的距离。

现在标题栏布局已经编写完成了，剩下的就是如何在程序中使用这个标题栏了，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" >

    <include layout="@layout/title" />

</LinearLayout>
```

没错！我们只需要通过一行 include 语句将标题栏布局引入进来就可以了。

最后别忘了在 MainActivity 中将系统自带的标题栏隐藏掉，代码如下所示：

```
public class MainActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        requestWindowFeature(Window.FEATURE_NO_TITLE);
        setContentView(R.layout.activity_main);
    }

}
```

现在运行一下程序，效果如图 3.27 所示。

图　3.27

使用这种方式，不管有多少布局需要添加标题栏，只需一行 include 语句就可以了。

3.4.2　创建自定义控件

引入布局的技巧确实解决了重复编写布局代码的问题，但是如果布局中有一些控件要求能够响应事件，我们还是需要在每个活动中为这些控件单独编写一次事件注册的代码。比如说标题栏中的返回按钮，其实不管是在哪一个活动中，这个按钮的功能都是相同的，即销毁掉当前活动。而如果在每一个活动中都需要重新注册一遍返回按钮的点击事件，无疑又是增加了很多重复代码，这种情况最好是使用自定义控件的方式来解决。

新建 TitleLayout 继承自 LinearLayout，让它成为我们自定义的标题栏控件，代码如下所示：

```
public class TitleLayout extends LinearLayout {

    public TitleLayout(Context context, AttributeSet attrs) {
        super(context, attrs);
        LayoutInflater.from(context).inflate(R.layout.title, this);
    }

}
```

首先我们重写了 LinearLayout 中的带有两个参数的构造函数，在布局中引入 TitleLayout 控件就会调用这个构造函数。然后在构造函数中需要对标题栏布局进行动态加载，这就要借助 LayoutInflater 来实现了。通过 LayoutInflater 的 from()方法可以构建出一个 LayoutInflater 对象，然后调用 inflate()方法就可以动态加载一个布局文件，inflate()方法接收两个参数，第一个参数是要加载的布局文件的 id，这里我们传入 R.layout.title，第二个参数是给加载好的布局再添加一个父布局，这里我们想要指定为 TitleLayout，于是直接传入 this。

现在自定义控件已经创建好了，然后我们需要在布局文件中添加这个自定义控件，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" >

    <com.example.uicustomviews.TitleLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        ></com.example.uicustomviews.TitleLayout>

</LinearLayout>
```

添加自定义控件和添加普通控件的方式基本是一样的，只不过在添加自定义控件的时候我们需要指明控件的完整类名，包名在这里是不可以省略的。

重新运行程序，你会发现此时效果和使用引入布局方式的效果是一样的。

然后我们来尝试为标题栏中的按钮注册点击事件，修改 TitleLayout 中的代码，如下所示：

```
public class TitleLayout extends LinearLayout {

    public TitleLayout(Context context, AttributeSet attrs) {
        super(context, attrs);
        LayoutInflater.from(context).inflate(R.layout.title, this);
        Button titleBack = (Button) findViewById(R.id.title_back);
        Button titleEdit = (Button) findViewById(R.id.title_edit);
        titleBack.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
                ((Activity) getContext()).finish();
            }
        });
        titleEdit.setOnClickListener(new OnClickListener() {
            @Override
```

```
            public void onClick(View v) {
                Toast.makeText(getContext(), "You clicked Edit button",
                        Toast.LENGTH_SHORT).show();
            }
        });
    }

}
```

首先还是通过 findViewById()方法得到按钮的实例，然后分别调用 setOnClickListener()方法给两个按钮注册了点击事件，当点击返回按钮时销毁掉当前的活动，当点击编辑按钮时弹出一段文本。重新运行程序，点击一下编辑按钮，效果如图 3.28 所示。

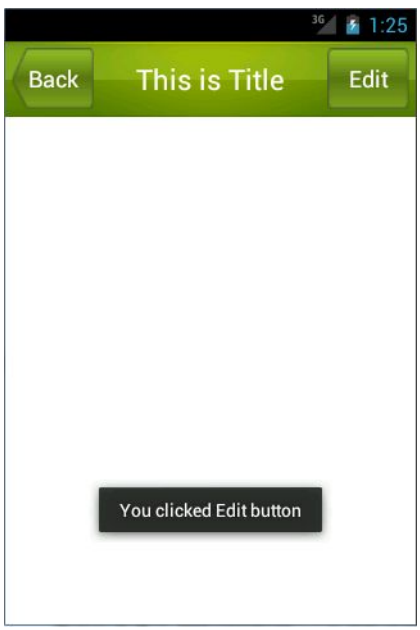

图　3.28

这样的话，每当我们在一个布局中引入 TitleLayout，返回按钮和编辑按钮的点击事件就已经自动实现好了，也是省去了很多编写重复代码的工作。

经验值：+1000　　目前经验值：5705

级别：菜鸟

赢得宝物：战胜自定义控件巨人。拾取自定义控件巨人掉落的宝物，只有一把灰色的小钥匙，很明显是开启什么用的。巨人已经跑掉了，我疑惑地顺着巨人深深的足迹寻到巨人的

家，小钥匙原来是房门钥匙，房子很小，里面除了一张床、一把椅子、一张桌子（桌子上放着一套积木、几张纸，还有两只笔），还有一个很小但很暖和的小炉子外，几乎空空如也。别问我为什么要闯入别人的家，游戏中不都是这样干的么，在神界的 Android 开发之旅中我也只是一个普通人。有时好像就是这样，你历尽千辛万苦，得到的却少得可怜。我把小钥匙插在锁上，离开了巨人的家。正准备继续前进，却发现前面山势陡然而起，峰峦耸入云霄，莫不是要遇到什么猛兽，我提了提手中的哨棒（谁刚才往我手中塞了根哨棒啊，莫不是雷锋？）容不得我细思量，天色不早，服下仅有的一颗大型信心增强大力丸。向山峦前进。

3.5 最常用和最难用的控件——ListView

ListView 绝对可以称得上是 Android 中最常用的控件之一，几乎所有的应用程序都会用到它。由于手机屏幕空间都比较有限，能够一次性在屏幕上显示的内容并不多，当我们的程序中有大量的数据需要展示的时候，就可以借助 ListView 来实现。ListView 允许用户通过手指上下滑动的方式将屏幕外的数据滚动到屏幕内，同时屏幕上原有的数据则会滚动出屏幕。相信你其实每天都在使用这个控件，比如查看手机联系人列表，翻阅微博的最新消息等等。

不过比起前面介绍的几种控件，ListView 的用法也相对复杂了很多，因此我们就单独使用一节内容来对 ListView 进行非常详细的讲解。

3.5.1 ListView 的简单用法

首先新建一个 ListViewTest 项目，并让 ADT 自动帮我们创建好活动。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" >

    <ListView
        android:id="@+id/list_view"
        android:layout_width="match_parent"
        android:layout_height="match_parent" >
    </ListView>

</LinearLayout>
```

在布局中加入 ListView 控件还算非常简单，先为 ListView 指定了一个 id，然后将宽度和高度都设置为 match_parent，这样 ListView 也就占据了整个布局的空间。

接下来修改 MainActivity 中的代码，如下所示：

```
public class MainActivity extends Activity {

    private String[] data = { "Apple", "Banana", "Orange", "Watermelon",
            "Pear", "Grape", "Pineapple", "Strawberry", "Cherry", "Mango" };

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        ArrayAdapter<String> adapter = new ArrayAdapter<String>(
                MainActivity.this, android.R.layout.simple_list_item_1, data);
        ListView listView = (ListView) findViewById(R.id.list_view);
        listView.setAdapter(adapter);
    }

}
```

既然 ListView 是用于展示大量数据的，那我们就应该先将数据提供好。这些数据可以是从网上下载的，也可以是从数据库中读取的，应该视具体的应用程序场景来决定。这里我们就简单使用了一个 data 数组来测试，里面包含了很多水果的名称。

不过，数组中的数据是无法直接传递给 ListView 的，我们还需要借助适配器来完成。Android 中提供了很多适配器的实现类，其中我认为最好用的就是 ArrayAdapter。它可以通过泛型来指定要适配的数据类型，然后在构造函数中把要适配的数据传入即可。ArrayAdapter 有多个构造函数的重载，你应该根据实际情况选择最合适的一种。这里由于我们提供的数据都是字符串，因此将 ArrayAdapter 的泛型指定为 String，然后在 ArrayAdapter 的构造函数中依次传入当前上下文、ListView 子项布局的 id，以及要适配的数据。注意我们使用了 android.R.layout.simple_list_item_1 作为 ListView 子项布局的 id，这是一个 Android 内置的布局文件，里面只有一个 TextView，可用于简单地显示一段文本。这样适配器对象就构建好了。

最后，还需要调用 ListView 的 setAdapter()方法，将构建好的适配器对象传递进去，这样 ListView 和数据之间的关联就建立完成了。

现在运行一下程序，效果如图 3.29 所示。

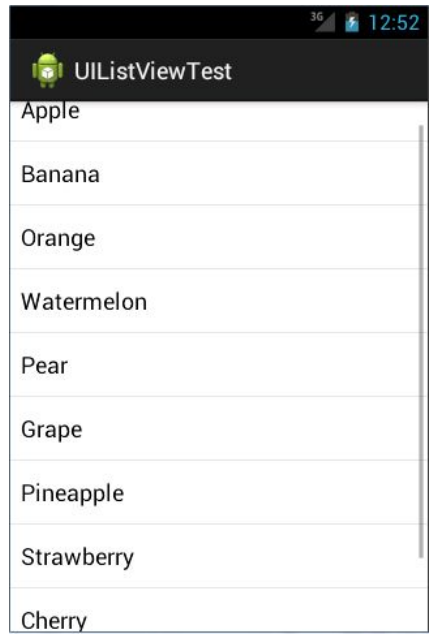

图 3.29

可以通过滚动的方式来查看屏幕外的数据。

3.5.2 定制 ListView 的界面

只能显示一段文本的 ListView 实在是太单调了，我们现在就来对 ListView 的界面进行定制，让它可以显示更加丰富的内容。

首先需要准备好一组图片，分别对应上面提供的每一种水果，待会我们要让这些水果名称的旁边都有一个图样。

接着定义一个实体类，作为 ListView 适配器的适配类型。新建类 Fruit，代码如下所示：

```
public class Fruit {

    private String name;

    private int imageId;

    public Fruit(String name, int imageId) {
        this.name = name;
        this.imageId = imageId;
    }
```

```
    public String getName() {
        return name;
    }

    public int getImageId() {
        return imageId;
    }

}
```

Fruit 类中只有两个字段，name 表示水果的名字，imageId 表示水果对应图片的资源 id。

然后需要为 ListView 的子项指定一个我们自定义的布局，在 layout 目录下新建 fruit_item.xml，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" >

    <ImageView
        android:id="@+id/fruit_image"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />

    <TextView
        android:id="@+id/fruit_name"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:layout_marginLeft="10dip" />

</LinearLayout>
```

在这个布局中，我们定义了一个 ImageView 用于显示水果的图片，又定义了一个 TextView 用于显示水果的名称。

接下来需要创建一个自定义的适配器，这个适配器继承自 ArrayAdapter，并将泛型指定为 Fruit 类。新建类 FruitAdapter，代码如下所示：

```
public class FruitAdapter extends ArrayAdapter<Fruit> {

    private int resourceId;
```

```
    public FruitAdapter(Context context, int textViewResourceId,
            List<Fruit> objects) {
        super(context, textViewResourceId, objects);
        resourceId = textViewResourceId;
    }

    @Override
    public View getView(int position, View convertView, ViewGroup parent) {
        Fruit fruit = getItem(position); // 获取当前项的Fruit实例
        View view = LayoutInflater.from(getContext()).inflate(resourceId, null);
        ImageView fruitImage = (ImageView) view.findViewById(R.id.fruit_image);
        TextView fruitName = (TextView) view.findViewById(R.id.fruit_name);
        fruitImage.setImageResource(fruit.getImageId());
        fruitName.setText(fruit.getName());
        return view;
    }

}
```

FruitAdapter 重写了父类的一组构造函数，用于将上下文、ListView 子项布局的 id 和数据都传递进来。另外又重写了 getView()方法，这个方法在每个子项被滚动到屏幕内的时候会被调用。在 getView 方法中，首先通过 getItem()方法得到当前项的 Fruit 实例，然后使用 LayoutInflater 来为这个子项加载我们传入的布局，接着调用 View 的 findViewById()方法分别获取到 ImageView 和 TextView 的实例，并分别调用它们的 setImageResource()和 setText()方法来设置显示的图片和文字，最后将布局返回，这样我们自定义的适配器就完成了。

下面修改 MainActivity 中的代码，如下所示：

```
public class MainActivity extends Activity {

    private List<Fruit> fruitList = new ArrayList<Fruit>();

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        initFruits(); // 初始化水果数据
        FruitAdapter adapter = new FruitAdapter(MainActivity.this,
R.layout.fruit_item, fruitList);
        ListView listView = (ListView) findViewById(R.id.list_view);
```

```
        listView.setAdapter(adapter);
    }

    private void initFruits() {
        Fruit apple = new Fruit("Apple", R.drawable.apple_pic);
        fruitList.add(apple);
        Fruit banana = new Fruit("Banana", R.drawable.banana_pic);
        fruitList.add(banana);
        Fruit orange = new Fruit("Orange", R.drawable.orange_pic);
        fruitList.add(orange);
        Fruit watermelon = new Fruit("Watermelon", R.drawable.watermelon_pic);
        fruitList.add(watermelon);
        Fruit pear = new Fruit("Pear", R.drawable.pear_pic);
        fruitList.add(pear);
        Fruit grape = new Fruit("Grape", R.drawable.grape_pic);
        fruitList.add(grape);
        Fruit pineapple = new Fruit("Pineapple", R.drawable.pineapple_pic);
        fruitList.add(pineapple);
        Fruit strawberry = new Fruit("Strawberry", R.drawable.strawberry_pic);
        fruitList.add(strawberry);
        Fruit cherry = new Fruit("Cherry", R.drawable.cherry_pic);
        fruitList.add(cherry);
        Fruit mango = new Fruit("Mango", R.drawable.mango_pic);
        fruitList.add(mango);
    }

}
```

可以看到，这里添加了一个 initFruits()方法，用于初始化所有的水果数据。在 Fruit 类的构造函数中将水果的名字和对应的图片 id 传入，然后把创建好的对象添加到水果列表中。接着我们在 onCreate()方法中创建了 FruitAdapter 对象，并将 FruitAdapter 作为适配器传递给了 ListView。这样定制 ListView 界面的任务就完成了。

现在重新运行程序，效果如图 3.30 所示。

图 3.30

虽然目前我们定制的界面还是很简单，但是相信聪明的你已经领悟到了诀窍，只要修改 fruit_item.xml 中的内容，就可以定制出各种复杂的界面了。

3.5.3 提升 ListView 的运行效率

之所以说 ListView 这个控件很难用，就是因为它有很多的细节可以优化，其中运行效率就是很重要的一点。目前我们 ListView 的运行效率是很低的，因为在 FruitAdapter 的 getView()方法中每次都将布局重新加载了一遍，当 ListView 快速滚动的时候这就会成为性能的瓶颈。

仔细观察，getView()方法中还有一个 convertView 参数，这个参数用于将之前加载好的布局进行缓存，以便之后可以进行重用。修改 FruitAdapter 中的代码，如下所示：

```
public class FruitAdapter extends ArrayAdapter<Fruit> {
    ……
    @Override
    public View getView(int position, View convertView, ViewGroup parent) {
        Fruit fruit = getItem(position);
        View view;
        if (convertView == null) {
            view = LayoutInflater.from(getContext()).inflate(resourceId, null);
        } else {
```

```
            view = convertView;
        }
        ImageView fruitImage = (ImageView) view.findViewById(R.id.fruit_image);
        TextView fruitName = (TextView) view.findViewById(R.id.fruit_name);
        fruitImage.setImageResource(fruit.getImageId());
        fruitName.setText(fruit.getName());
        return view;
    }
}
```

可以看到，现在我们在 getView()方法中进行了判断，如果 convertView 为空，则使用 LayoutInflater 去加载布局，如果不为空则直接对 convertView 进行重用。这样就大大提高了 ListView 的运行效率，在快速滚动的时候也可以表现出更好的性能。

不过，目前我们的这份代码还是可以继续优化的，虽然现在已经不会再重复去加载布局，但是每次在 getView()方法中还是会调用 View 的 findViewById()方法来获取一次控件的实例。我们可以借助一个 ViewHolder 来对这部分性能进行优化，修改 FruitAdapter 中的代码，如下所示：

```
public class FruitAdapter extends ArrayAdapter<Fruit> {
    ……
    @Override
    public View getView(int position, View convertView, ViewGroup parent) {
        Fruit fruit = getItem(position);
        View view;
        ViewHolder viewHolder;
        if (convertView == null) {
            view = LayoutInflater.from(getContext()).inflate(resourceId, null);
            viewHolder = new ViewHolder();
            viewHolder.fruitImage = (ImageView) view.findViewById
(R.id.fruit_image);
            viewHolder.fruitName = (TextView) view.findViewById
(R.id.fruit_name);
            view.setTag(viewHolder); // 将ViewHolder存储在View中
        } else {
            view = convertView;
            viewHolder = (ViewHolder) view.getTag(); // 重新获取ViewHolder
        }
        viewHolder.fruitImage.setImageResource(fruit.getImageId());
        viewHolder.fruitName.setText(fruit.getName());
        return view;
```

```
    }

    class ViewHolder {

        ImageView fruitImage;

        TextView fruitName;

    }
}
```

我们新增了一个内部类 ViewHolder，用于对控件的实例进行缓存。当 convertView 为空的时候，创建一个 ViewHolder 对象，并将控件的实例都存放在 ViewHolder 里，然后调用 View 的 setTag()方法，将 ViewHolder 对象存储在 View 中。当 convertView 不为空的时候则调用 View 的 getTag()方法，把 ViewHolder 重新取出。这样所有控件的实例都缓存在了 ViewHolder 里，就没有必要每次都通过 findViewById()方法来获取控件实例了。

通过这两步的优化之后，我们 ListView 的运行效率就已经非常不错了。

3.5.4 ListView 的点击事件

话说回来，ListView 的滚动毕竟只是满足了我们视觉上的效果，可是如果 ListView 中的子项不能点击的话，这个控件就没有什么实际的用途了。因此，本小节中我们就来学习一下 ListView 如何才能响应用户的点击事件。

修改 MainActivity 中的代码，如下所示：

```
public class MainActivity extends Activity {

    private List<Fruit> fruitList = new ArrayList<Fruit>();

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        initFruits();
        FruitAdapter adapter = new FruitAdapter(MainActivity.this,
R.layout.fruit_item, fruitList);
        ListView listView = (ListView) findViewById(R.id.list_view);
        listView.setAdapter(adapter);
        listView.setOnItemClickListener(new OnItemClickListener() {
            @Override
```

```
            public void onItemClick(AdapterView<?> parent, View view,
                    int position, long id) {
                Fruit fruit = fruitList.get(position);
                Toast.makeText(MainActivity.this, fruit.getName(),
                        Toast.LENGTH_SHORT).show();
            }
        });
    }
    ......
}
```

可以看到，我们使用了 setOnItemClickListener()方法来为 ListView 注册了一个监听器，当用户点击了 ListView 中的任何一个子项时就会回调 onItemClick()方法，在这个方法中可以通过 position 参数判断出用户点击的是哪一个子项，然后获取到相应的水果，并通过 Toast 将水果的名字显示出来。

重新运行程序，并点击一下西瓜，效果如图 3.31 所示。

图　3.31

经验值：+1200　　目前经验值：6905

级别：菜鸟

赢得宝物：战胜 ListView 半神。拾取 ListView 半神掉落的宝物，神人魔三界护照、大数

据列表显示咒语。ListView 是神界为数不多的半神，作为少数民族，ListView 获得的尊重和优惠条件却比神族还多，比如他就成功申请到了很多神族梦寐以求的神人魔三界护照，可以自由出入三界，他的主业是在三界种植和贩卖时鲜水果，良心生意，绝不使用化肥和激素，业余喜欢写写 Android 应用。

3.6 单位和尺寸

前面我们说过，为了要让程序拥有更好的屏幕适配能力，在指定控件和布局大小的时候最好使用 match_parent 和 wrap_content，尽量避免将控件的宽和高设定一个固定值。不过在有些情况下，仅仅使用 match_parent 和 wrap_content 确实无法满足我们的需求，这时就必须要给控件的宽或高指定一个固定值。在布局文件中指定宽高的固定大小有以下常用单位可供选择：px、pt、dp 和 sp。新建好一个 UISizeTest 项目，然后我们开始对这几个单位进行探讨。

3.6.1 px 和 pt 的窘境

px 是像素的意思，即屏幕中可以显示的最小元素单元，我们应用里任何可见的东西都是由一个个像素点组成的。单独一个像素点非常的微小，肉眼是无法看见的，可是当许许多多的像素点聚集到一起时，就可以拼接成五彩缤纷的图案。

pt 是磅数的意思，1 磅等于 1/72 英寸，一般 pt 都会作为字体的单位来使用。

过去在 PC 上使用 px 和 pt 的时候可以说是非常得心应手，能把程序打扮得漂漂亮亮。可是现在到了手机上，这两个单位就显得有些力不从心了，因为手机的分辨率各不相同，一个 200px 宽的按钮在低分辨率的手机上可能将近占据满屏，而到了高分辨率的手机上可能只占据屏幕的一半。我们通过例子来直观地看一下，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/button"
        android:layout_width="200px"
        android:layout_height="wrap_content"
        android:text="Button"
        />

</LinearLayout>
```

这里通过 android:layout_width 属性将按钮的宽指定为 200px，然后运行程序，效果如图 3.32 所示。

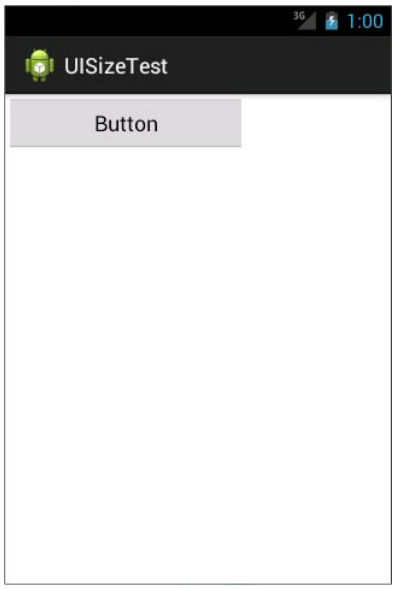

图　3.32

接着创建一个 240*320 像素的低分辨率模拟器，在这个模拟器上重新运行程序，效果如图 3.33 所示。

图　3.33

可以明显看出，同样 200px 宽的按钮在不同分辨率的屏幕上显示的效果是完全不同的，pt 的情况和 px 差不多，这导致这两个单位在手机领域上面很难有所发挥。

3.6.2 dp 和 sp 来帮忙

谷歌当然也意识到了这个令人头疼了问题，于是为 Android 引入了一套新的单位 dp 和 sp。

dp 是密度无关像素的意思，也被称作 dip，和 px 相比，它在不同密度的屏幕中的显示比例将保持一致。

sp 是可伸缩像素的意思，它采用了和 dp 同样的设计理念，解决了文字大小的适配问题。

这里有一个新名词需要引起我们的注意，什么叫密度？Android 中的密度就是屏幕每英寸所包含的像素数，通常以 dpi 为单位。比如一个手机屏幕的宽是 2 英寸长是 3 英寸，如果它的分辨率是 320*480 像素，那这个屏幕的密度就是 160dpi，如果它的分辨率是 640*960，那这个屏幕的密度就是 320dpi，因此密度值越高的屏幕显示的效果就越精细。我们可以通过代码来得知当前屏幕的密度值是多少，修改 MainActivity 中的代码，如下所示：

```
public class MainActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        float xdpi = getResources().getDisplayMetrics().xdpi;
        float ydpi = getResources().getDisplayMetrics().ydpi;
        Log.d("MainActivity", "xdpi is " + xdpi);
        Log.d("MainActivity", "ydpi is " + ydpi);
    }

}
```

可以看到，在 onCreate()方法中我们动态获取到了当前屏幕的密度值，并打印出来，重新运行程序，结果如图 3.34 所示。

Tag	Text
MainActivity	xdpi is 160.0
MainActivity	ydpi is 160.0

图 3.34

然后在低分辨率的模拟器上重新运行程序，结果如图 3.35 所示。

Tag	Text
MainActivity	xdpi is 120.0
MainActivity	ydpi is 120.0

图　3.35

根据 Android 的规定，在 160dpi 的屏幕上，1dp 等于 1px，而在 320dpi 的屏幕上，1dp 就等于 2px。因此，使用 dp 来指定控件的宽和高，就可以保证控件在不同密度的屏幕中的显示比例保持一致。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/button"
        android:layout_width="200dp"
        android:layout_height="wrap_content"
        android:text="Button"
        />

</LinearLayout>
```

这里我们将按钮的宽度改成了 200dp，重新运行程序，效果如图 3.36 所示：

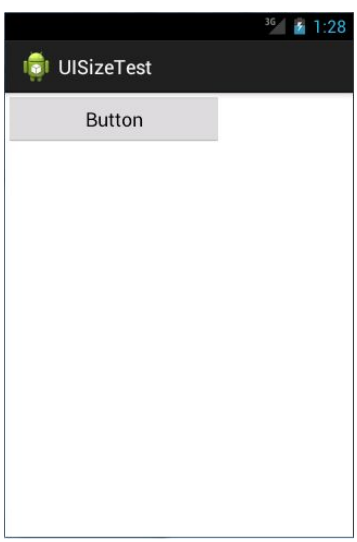

图　3.36

咦？怎么感觉和之前的宽度没有什么区别呢？这是因为我们模拟器的屏幕密度刚好是160dpi，这时的 1dp 就等于 1px。

然后在低分辨率的模拟器上重新运行程序，效果如图 3.37 所示。

图　3.37

这时就可以明显看出不同了吧！对比两个模拟器的运行结果，你会发现按钮在不同分辨率的屏幕上所占大小的比例几乎是相同的。sp 的原理和 dp 是一样的，它主要是用于指定文字的大小，这里就不再进行介绍了。

总结一下，在编写 Android 程序的时候，尽量将控件或布局的大小指定成 match_parent 或 wrap_content，如果必须要指定一个固定值，则使用 dp 来作为单位，指定文字大小的时候使用 sp 作为单位。

3.7　编写界面的最佳实践

既然已经学习了那么多 UI 开发的知识，也是时候应该实战一下了。这次我们要综合运用前面所学的大量内容来编写出一个较为复杂且相当美观的聊天界面，你准备好了吗？要先创建一个 UIBestPractice 项目才算准备好了哦。

3.7.1　制作 Nine-Patch 图片

在实战正式开始之前，我们还需要先学习一下如何制作 Nine-Patch 图片。你可能之前还没有听说过这个名词，它是一种被特殊处理过的 png 图片，能够指定哪些区域可以被拉伸而哪些区域不可以。

那么 Nine-Patch 图片到底有什么实际作用呢？我们还是通过一个例子来看一下吧。比如说项目中有一张气泡样式的图片 message_left.png，如图 3.38 所示。

图　3.38

我们将这张图片设置为一个 LinearLayout 的背景图片，修改 activity_main.xml 中的代码，如下所示：

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

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:background="@drawable/message_left" >
    </LinearLayout>

</RelativeLayout>
```

将 LinearLayout 的宽度指定为 match_parent，然后将它的背景图设置为 message_left，现在运行程序，效果如图 3.39 所示。

图　3.39

可以看到，由于 message_left 的宽度不足以填满整个屏幕的宽度，整张图片被均匀地拉伸了！这种效果非常差，用户肯定是不能容忍的，这时我们就可以使用 Nine-Patch 图片来进行改善。

在 Android sdk 目录下有一个 tools 文件夹，在这个文件夹中找到 draw9patch.bat 文件，我们就是使用它来制作 Nine-Patch 图片的。双击打开之后，在导航栏点击 File→Open 9-patch 将 message_left.png 加载进来，如图 3.40 所示。

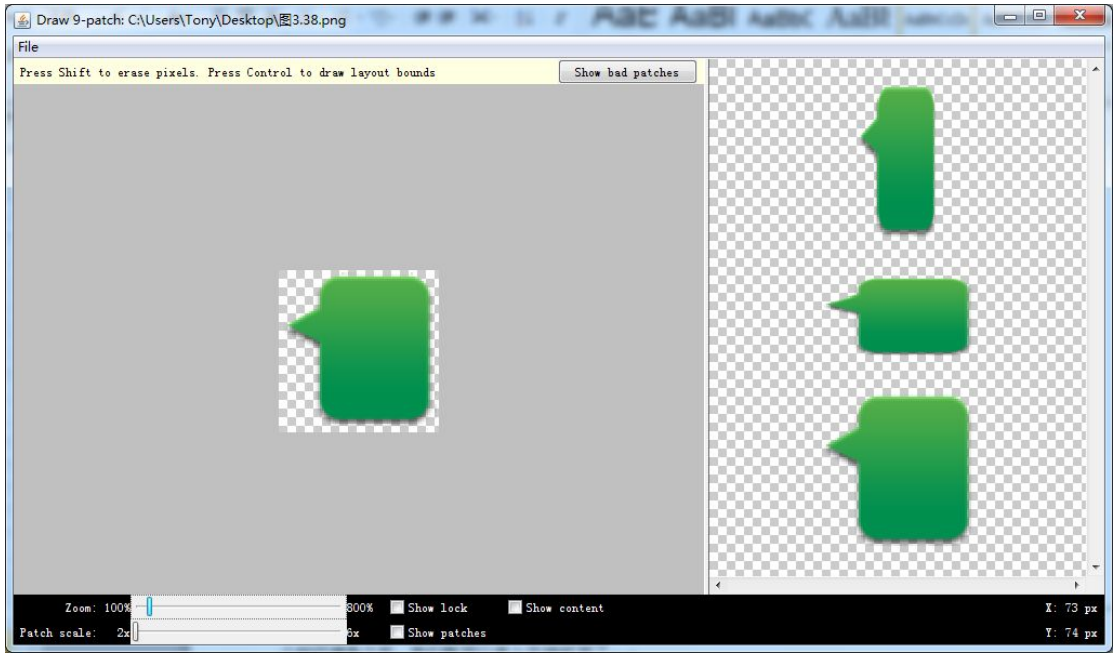

图　3.40

我们可以在图片的四个边框绘制一个个的小黑点，在上边框和左边框绘制的部分就表示当图片需要拉伸时就拉伸黑点标记的区域，在下边框和右边框绘制的部分则表示内容会被放置的区域。绘制完成后效果如图 3.41 所示。

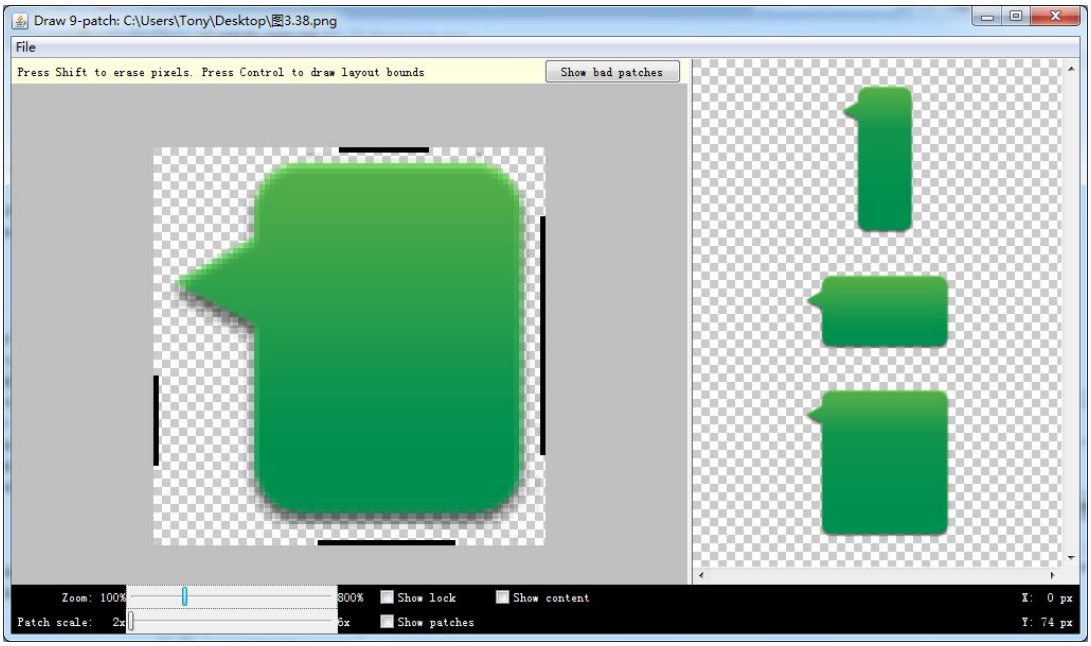

图　3.41

最后点击导航栏 File→Save 9-patch 把绘制好的图片进行保存，此时的文件名就是 message_left.9.png。

使用这张图片替换掉之前的 message_left.png 图片，重新运行程序，效果如图 3.42 所示。

图　3.42

这样当图片需要拉伸的时候，就可以只拉伸指定的区域，程序在外观上也是有了很大的改进。有了这个知识储备之后，我们就可以进入实战环节了。

3.7.2 编写精美的聊天界面

既然是要编写一个聊天界面，那就肯定要有收到的消息和发出的消息。上一节中我们制作的 message_left.9.png 可以作为收到消息的背景图，那么毫无疑问你还需要再制作一张 message_right.9.png 作为发出消息的背景图。

图片都提供好了之后就可以开始编码了，首先还是编写主界面，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:background="#d8e0e8"
    android:orientation="vertical" >

    <ListView
        android:id="@+id/msg_list_view"
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="1"
        android:divider="#0000" >
    </ListView>

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

        <EditText
            android:id="@+id/input_text"
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:hint="Type somthing here"
            android:maxLines="2" />

        <Button
            android:id="@+id/send"
            android:layout_width="wrap_content"
```

```
            android:layout_height="wrap_content"
            android:text="Send" />

    </LinearLayout>

</LinearLayout>
```

这里在主界面中放置了一个 ListView 用于显示聊天的消息内容，又放置了一个 EditText 用于输入消息，还放置了一个 Button 用于发送消息。ListView 中用到了一个 android:divider 属性，它可以指定 ListView 分隔线的颜色，这里#0000 表示将分隔线设为透明色。其他用到的所有属性都是我们之前学过的，相信你理解起来应该不费力。

然后我们来定义消息的实体类，新建 Msg，代码如下所示：

```
public class Msg {

    public static final int TYPE_RECEIVED = 0;

    public static final int TYPE_SENT = 1;

    private String content;

    private int type;

    public Msg(String content, int type) {
        this.content = content;
        this.type = type;
    }

    public String getContent() {
        return content;
    }

    public int getType() {
        return type;
    }

}
```

Msg 类中只有两个字段，content 表示消息的内容，type 表示消息的类型。其中消息类型有两个值可选，TYPE_RECEIVED 表示这是一条收到的消息，TYPE_SENT 表示这是一条发出的消息。

接着来编写 ListView 子项的布局，新建 msg_item.xml，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"
    android:padding="10dp" >

    <LinearLayout
        android:id="@+id/left_layout"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="left"
        android:background="@drawable/message_left" >

        <TextView
            android:id="@+id/left_msg"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_gravity="center"
            android:layout_margin="10dp"
            android:textColor="#fff" />

    </LinearLayout>

    <LinearLayout
        android:id="@+id/right_layout"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="right"
        android:background="@drawable/message_right" >

        <TextView
            android:id="@+id/right_msg"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_gravity="center"
            android:layout_margin="10dp" />

    </LinearLayout>

</LinearLayout>
```

这里我们让收到的消息居左对齐，发出的消息居右对齐，并且分别使用 message_left.9.png 和 message_right.9.png 作为背景图。你可能会有些疑虑，怎么能让收到的消息和发出的消息都放在同一个布局里呢？不用担心，还记得我们前面学过的可见属性吗，只要稍后在代码中根据消息的类型来决定隐藏和显示哪种消息就可以了。

接下来需要创建 ListView 的适配器类，让它继承自 ArrayAdapter，并将泛型指定为 Msg 类。新建类 MsgAdapter，代码如下所示：

```
public class MsgAdapter extends ArrayAdapter<Msg> {

    private int resourceId;

    public MsgAdapter(Context context, int textViewResourceId, List<Msg>
objects) {
        super(context, textViewResourceId, objects);
        resourceId = textViewResourceId;
    }

    @Override
    public View getView(int position, View convertView, ViewGroup parent) {
        Msg msg = getItem(position);
        View view;
        ViewHolder viewHolder;
        if (convertView == null) {
            view = LayoutInflater.from(getContext()).inflate(resourceId, null);
            viewHolder = new ViewHolder();
            viewHolder.leftLayout = (LinearLayout) view.findViewById
(R.id.left_layout);
            viewHolder.rightLayout = (LinearLayout) view.findViewById
(R.id.right_layout);
            viewHolder.leftMsg = (TextView) view.findViewById(R.id.left_msg);
            viewHolder.rightMsg = (TextView) view.findViewById(R.id.right_msg);
            view.setTag(viewHolder);
        } else {
            view = convertView;
            viewHolder = (ViewHolder) view.getTag();
        }
        if (msg.getType() == Msg.TYPE_RECEIVED) {
            // 如果是收到的消息，则显示左边的消息布局，将右边的消息布局隐藏
            viewHolder.leftLayout.setVisibility(View.VISIBLE);
```

```
            viewHolder.rightLayout.setVisibility(View.GONE);
            viewHolder.leftMsg.setText(msg.getContent());
        } else if(msg.getType() == Msg.TYPE_SENT) {
            // 如果是发出的消息，则显示右边的消息布局，将左边的消息布局隐藏
            viewHolder.rightLayout.setVisibility(View.VISIBLE);
            viewHolder.leftLayout.setVisibility(View.GONE);
            viewHolder.rightMsg.setText(msg.getContent());
        }
        return view;
    }

    class ViewHolder {

        LinearLayout leftLayout;

        LinearLayout rightLayout;

        TextView leftMsg;

        TextView rightMsg;

    }

}
```

以上代码你应该是非常熟悉了，和我们学习 ListView 那一节的代码基本是一样的，只不过在 getView()方法中增加了对消息类型的判断。如果这条消息是收到的，则显示左边的消息布局，如果这条消息是发出的，则显示右边的消息布局。

最后修改 MainActivity 中的代码，来为 ListView 初始化一些数据，并给发送按钮加入事件响应，代码如下所示：

```
public class MainActivity extends Activity {

    private ListView msgListView;

    private EditText inputText;

    private Button send;

    private MsgAdapter adapter;
```

```
    private List<Msg> msgList = new ArrayList<Msg>();

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        requestWindowFeature(Window.FEATURE_NO_TITLE);
        setContentView(R.layout.activity_main);
        initMsgs(); // 初始化消息数据
        adapter = new MsgAdapter(MainActivity.this, R.layout.msg_item, msgList);
        inputText = (EditText) findViewById(R.id.input_text);
        send = (Button) findViewById(R.id.send);
        msgListView = (ListView) findViewById(R.id.msg_list_view);
        msgListView.setAdapter(adapter);
        send.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
                String content = inputText.getText().toString();
                if (!"".equals(content)) {
                    Msg msg = new Msg(content, Msg.TYPE_SENT);
                    msgList.add(msg);
                    adapter.notifyDataSetChanged(); // 当有新消息时，刷新
ListView中的显示
                    msgListView.setSelection(msgList.size()); // 将ListView
定位到最后一行
                    inputText.setText(""); // 清空输入框中的内容
                }
            }
        });
    }

    private void initMsgs() {
        Msg msg1 = new Msg("Hello guy.", Msg.TYPE_RECEIVED);
        msgList.add(msg1);
        Msg msg2 = new Msg("Hello. Who is that?", Msg.TYPE_SENT);
        msgList.add(msg2);
        Msg msg3 = new Msg("This is Tom. Nice talking to you. ", Msg.TYPE_RECEIVED);
        msgList.add(msg3);
    }

}
```

在 initMsgs()方法中我们先初始化了几条数据用于在 ListView 中显示。然后在发送按钮的点击事件里获取了 EditText 中的内容，如果内容不为空则创建出一个新的 Msg 对象，并把它添加到 msgList 列表中去。之后又调用了适配器的 notifyDataSetChanged()方法，用于通知列表的数据发生了变化，这样新增的一条消息才能够在 ListView 中显示。接着调用 ListView 的 setSelection()方法将显示的数据定位到最后一行，以保证一定可以看得到最后发出的一条消息。最后调用 EditText 的 setText()方法将输入的内容清空。

这样所有的工作就都完成了，终于可以检验一下我们的成果了，运行程序之后你将会看到非常美观的聊天界面，并且可以输入和发送消息，如图 3.43 所示。

图 3.43

经过这个例子的实战之后，我相信不仅加深了你对本章中所学 UI 知识的理解，还让你有了如何灵活运用这些知识来设计出优秀界面的思路。这一章也是学了不少东西，让我们来总结一下吧。

3.8 小结与点评

虽然本章的内容很多，但我觉得学习起来应该还是挺愉快的吧。不同于上一章中我们来来回回使用那几个按钮，本章可以说是使用了各种各样的控件，制作出了丰富多彩的界面。尤其是在实战环节编写出了那么精美的聊天界面，你的满足感应该比上一章还要强吧？

本章从 Android 中的一些常见控件开始入手，依次介绍了基本布局的用法、自定义控件的方法、ListView 的详细用法以及 Android 中单位的选择和使用，基本已经将重要的 UI 知识点全部覆盖了。想想在开始的时候我说不推荐使用可视化的编辑工具，而是应该全部使用 XML 的方式来编写界面，现在你是不是已经感觉使用 XML 非常的简单了呢？并且在以后不管是面对多么复杂的界面，我希望你都能够自信满满，因为真正理解了界面编写的原理之后，是没有什么能够难倒你的。

不过到目前为止，我们都只是学习了 Android 手机方面的开发技巧，下一章中将会涉及一些 Android 平板方面的知识点，能够同时兼容手机和平板也是 Android 4.0 系统的新特性，适当地放松和休息一段时间后，我们再来继续前行吧！

经验值：+1500　　目前经验值：8405

级别：菜鸟

获赠宝物：在经过 UI 镇时，在皮匠铺的皮匠杰弗里·皮处获赠一部上古配色奇书，书名叫《色即是空》，作者是上古奇人有色氏。获赠的原因说来也巧，各位看官可能还记得，在战胜周期兽猎人后，我试枪时曾修复过一只 bug 鸟，而那只 bug 鸟正是这位杰弗里·皮最钟爱的宠物，前不久这鸟在野外玩耍时不慎被一只健全猫咬掉了一只翅膀，后使诈才侥幸逃脱，万幸被我一枪打回了原形，才得以和主人团聚。老皮为了感谢我，送了我这部奇书。说它奇是因为这本书至今无人能看懂，原因是这书本身是写在神界特有的黄山羊的羊皮上的，而黄山羊皮本身又黄得吓死人，以致在这么黄的表面上涂抹任何颜色都严重偏色，最终致使无人能看懂这部书中所表达的那些精彩绝伦的配色范例。至于为什么配色界的鼻祖有色氏会犯下这么低级的错误，学术界至今没有达成共识，学院派认为此事必有蹊跷、必有深意，只是今人无法揣测圣意，而阴谋论者则认为有色氏实际上是位患有色盲的人氏（简称有色氏），为向命运发起终极挑战而写下了这部巨著。而为什么这部极具研究价值的上古典籍的原本会流落到一个皮匠的家里，老皮表示他也不晓得，只知道是祖上传下来的。这书留在他这里也没多大用，不如送恩人，或许我这位人界的青年才俊有朝一日能破解此书的秘密。谢过，收好书。把一些无用的物资在镇里卖掉换了些盘缠，减轻一下负重。继续前进。