

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，不会像普通用户那样仅仅是关注一下它的界面和功能，而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逻辑来完成，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。你在上一章中反反复复地使用那几个按钮，想必都快要吐了吧，本章我们就来学习更多的 UI 开发方面的知识。

3.1 如何编写程序界面

Android 中有多种编写程序界面的方式可供选择。Android Studio 和 Eclipse 中都提供了相应的可视化编辑器，允许使用拖放控件的方式来编写布局，并能在视图上直接修改控件的属性。不过我并不推荐你使用这种方式来编写界面，因为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。通过这种方式制作出的界面通常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都将通过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。

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

3.2 常用控件的使用方法

Android 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 Android Studio 自动创建活动，活动名和布局名都使用默认值。

3.2.1 TextView

TextView 可以说是 Android 中最简单的一个控件了，你在前面其实已经和它打过一些交道了。它主要用于在界面上显示一段文本信息，比如你在第 1 章看到的“Hello world!”。下面我们就来看一看关于 TextView 的更多用法。

修改 activity_main.xml 中的代码，如下所示：

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

    <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` 和 `android:layout_height` 指定了控件的宽度和高度。Android 中所有的控件都具有这两个属性，可选值有 3 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 TextView 中显示的文本内容，现在运行程序，效果如图 3.1 所示。

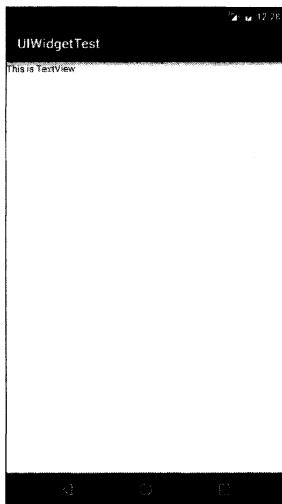


图 3.1 TextView 运行效果

虽然指定的文本内容正常显示了，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，可是由于文字内容不够长，所以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:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <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。

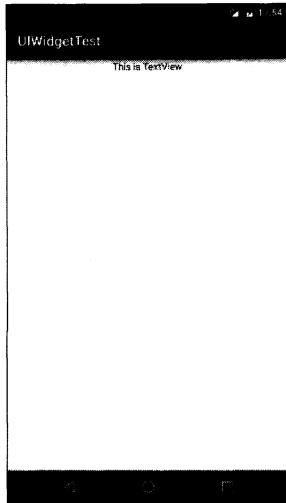


图 3.2 TextView 居中效果

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:orientation="vertical"  
    android:layout_width="match_parent"  
    android:layout_height="match_parent">  
  
    <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，在 Android 中字体大小使用 sp 作为单位。重新运行程序，效果如图 3.3 所示。

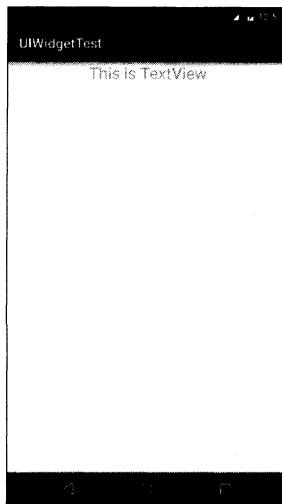


图 3.3 改变 TextView 文字大小和颜色的效果

当然 TextView 中还有很多其他的属性，这里就不再一一介绍了，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:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    ...

    <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。

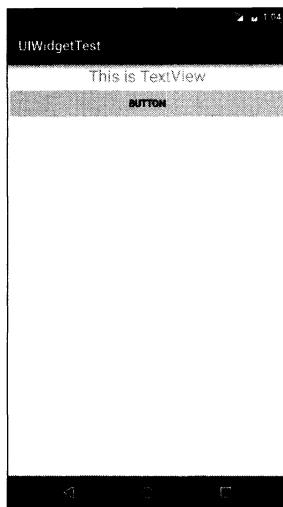


图 3.4 Button 运行效果

细心的你可能会留意到，我们在布局文件里面设置的文字是“Button”，但最终的显示结果却是“BUTTON”。这是由于系统会对 Button 中的所有英文字母自动进行大写转换，如果这不是你想要的效果，可以使用如下配置来禁用这一默认特性：

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

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

```
public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Button button = (Button) findViewById(R.id.button);
        button.setOnClickListener(new View.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 AppCompatActivity implements View.OnClickListener {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Button 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的喜好了。

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 的应用场景非常普遍，在进行发短信、发微博、聊 QQ 等操作时，你不得不使用 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:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    ...

    <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，用法基本上都很相似：给控件定义一个 id，再指定控件的宽度和高度，然后再适当加入一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。

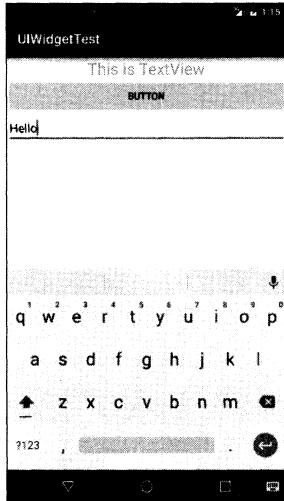


图 3.5 EditText 运行效果

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:orientation="vertical"  
    android:layout_width="match_parent"  
    android:layout_height="match_parent">  
  
    ...  
  
    <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。

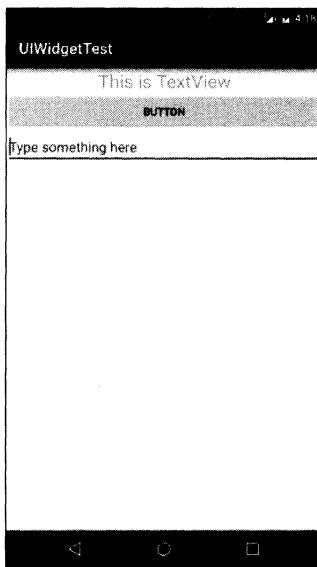


图 3.6 EditText 设置 hint 效果

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:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    ...
    <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。

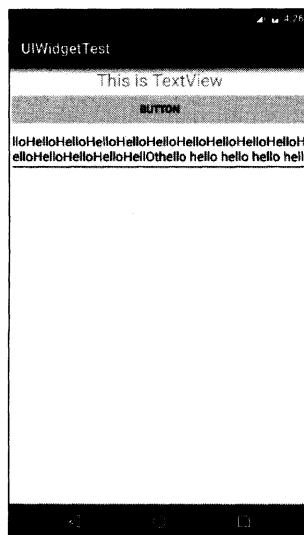


图 3.7 EditText 设置 maxLines 效果

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 AppCompatActivity implements View.OnClickListener {  
  
    private EditText editText;  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
        Button 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。

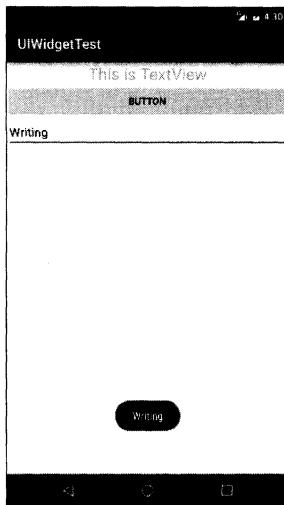


图 3.8 获取 `EditText` 中输入的内容

3.2.4 ImageView

`ImageView` 是用于在界面上展示图片的一个控件，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，图片通常都是放在以“`drawable`”开头的目录下的。目前我们的项目中有一个空的 `drawable` 目录，不过由于这个目录没有指定具体的分辨率，所以一般不使用它来放置图片。这里我们在 `res` 目录下新建一个 `drawable-xhdpi` 目录，然后将事先准备好的两张图片 `img_1.png` 和 `img_2.png` 复制到该目录当中。

接下来修改 `activity_main.xml`，如下所示：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"  
    android:orientation="vertical"  
    android:layout_width="match_parent"  
    android:layout_height="match_parent">  
  
    ...  
  
<ImageView  
    android:id="@+id/image_view"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:src="@drawable/img_1 "
```

```
/>

</LinearLayout>
```

可以看到，这里使用 `android:src` 属性给 `ImageView` 指定了一张图片。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`，这样就保证了不管图片的尺寸是多少，图片都可以完整地展示出来。重新运行程序，效果如图 3.9 所示。

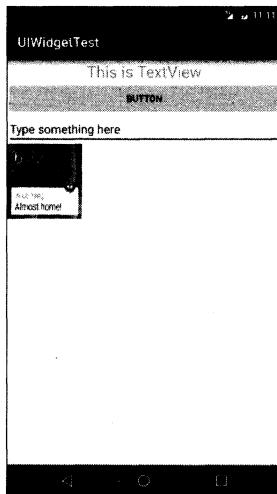


图 3.9 ImageView 运行效果

我们还可以在程序中通过代码动态地更改 `ImageView` 中的图片，然后修改 `MainActivity` 的代码，如下所示：

```
public class MainActivity extends AppCompatActivity implements View.OnClickListener {

    private EditText editText;
    private ImageView imageView;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Button 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.img_2);
        }
    }
}
```

```

        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 `img_2`，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。

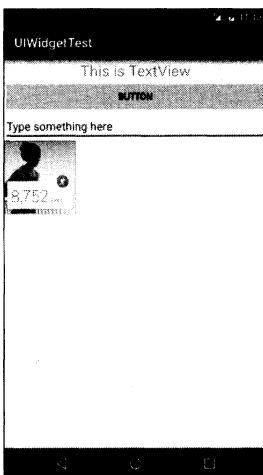


图 3.10 动态更改 ImageView 中的图片

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:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    ...

    <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。

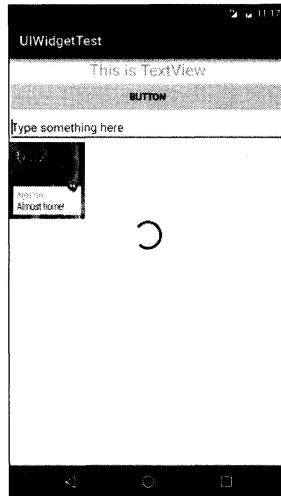


图 3.11 ProgressBar 运行效果

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 3 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` 这 3 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 AppCompatActivity implements View.OnClickListener {
    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 = (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:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    ...

    <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 AppCompatActivity implements View.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。

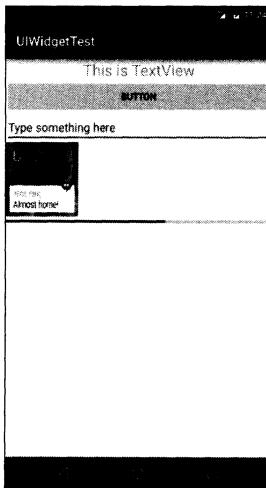


图 3.12 ProgressBar 水平样式效果

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 AlertDialog 一般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 AppCompatActivity implements View.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。

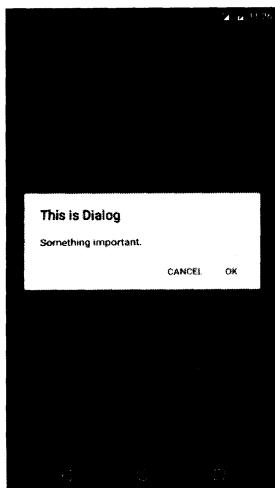


图 3.13 `AlertDialog` 运行效果

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 AppCompatActivity implements View.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。



图 3.14 ProgressDialog 运行效果

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。

好了，关于 Android 常用控件的使用，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，当本书后面涉及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。

3.3 详解 4 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，图 3.15 很好地展示了它们之间的关系。

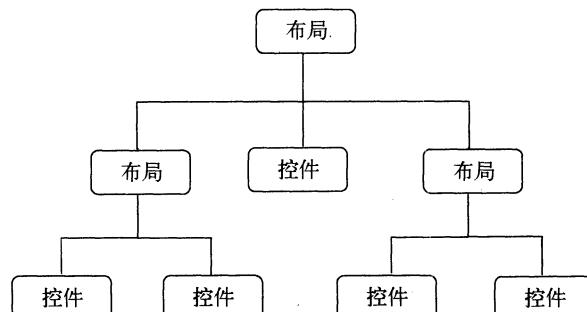


图 3.15 布局和控件的关系

下面我们来详细讲解下 Android 中 4 种最基本的布局。先做好准备工作，新建一个 `UILayoutTest` 项目，并让 Android Studio 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 线性布局

`LinearLayout` 又称作线性布局，是一种非常常用的布局。正如它的名字所描述的一样，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:orientation="vertical"
    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: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 3 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。

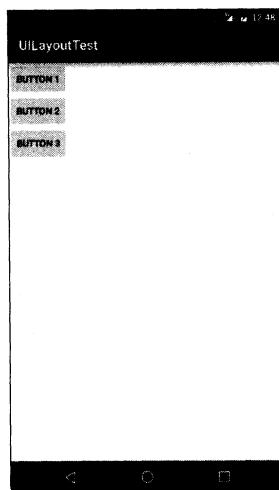


图 3.16 LinearLayout 垂直排列

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:orientation="horizontal"  
    android:layout_width="match_parent"  
    android:layout_height="match_parent">  
  
    ...  
  
</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。

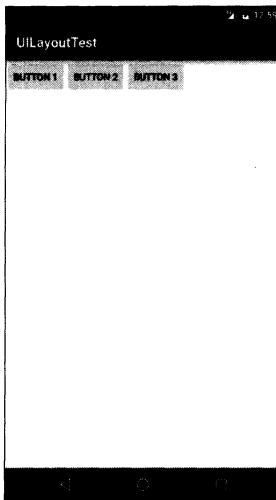


图 3.17 LinearLayout 水平排列

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`。

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？其实从名字就可以看出，`android:gravity` 用于指定文字在控件中的对齐方式，而 `android:layout_gravity` 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:orientation="horizontal"
    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_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。

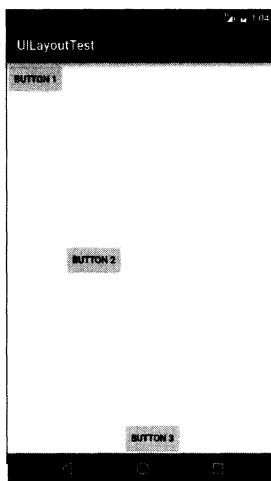


图 3.18 指定 layout_gravity 的效果

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:orientation="horizontal"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <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 0dp，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 0dp 是一种比较规范的写法。另外，dp 是 Android 中用于指定控件大小、间距等属性的单位，后面我们还会经常用到它。

然后在 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。

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。

重新运行程序，你会看到如图 3.19 所示的效果。

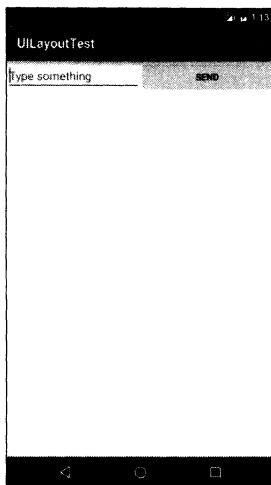


图 3.19 指定 `layout_weight` 的效果

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:orientation="horizontal"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <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。

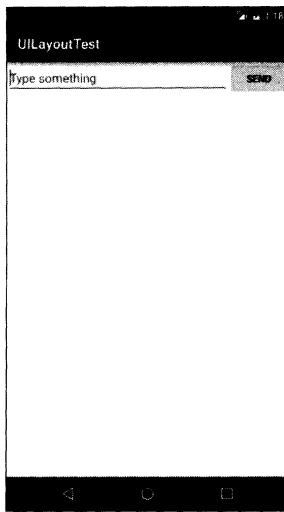


图 3.20 使用 `layout_weight` 实现宽度自适配效果

3.3.2 相对布局

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。

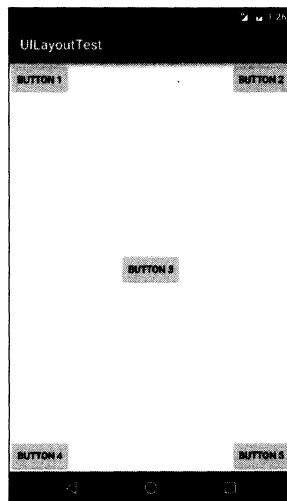


图 3.21 相对于父布局定位的效果

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

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。

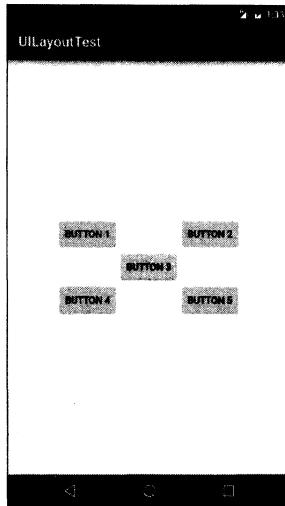


图 3.22 相对于控件定位的效果

`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。此外，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吧。

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 帧布局

`FrameLayout` 又称作帧布局，它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。这种布局没有方便的定位方式，所有的控件都会默认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">

    <TextView
        android:id="@+id/text_view"
        android:layout_width="wrap_content"
```

```
    android:layout_height="wrap_content"
    android:text="This is TextView"
  />

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

FrameLayout 中只是放置了一个 TextView 和一个 ImageView。需要注意的是，当前项目我们没有准备任何图片，所以这里 ImageView 直接使用了@mipmap 来访问 ic_launcher 这张图，虽说这种用法的场景可能非常少，但我还是要告诉你，这是完全可行的。重新运行程序，效果如图 3.23 所示。

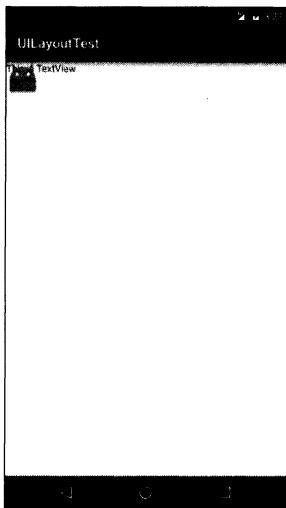


图 3.23 FrameLayout 运行效果

可以看到，文字和图片都是位于布局的左上角。由于 ImageView 是在 TextView 之后添加的，因此图片压在了文字的上面。

当然除了这种默认效果之外，我们还可以使用 `layout_gravity` 属性来指定控件在布局中的对齐方式，这和 LinearLayout 中的用法是相似的。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">

  <TextView
    android:id="@+id/text_view"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="top|left"
    android:text="This is TextView"/>
  <ImageView
    android:id="@+id/image_view"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:src="@mipmap/ic_launcher"/>
</FrameLayout>
```

```
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="left"
    android:text="This is TextView"
    />

<ImageView
    android:id="@+id/button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="right"
    android:src="@mipmap/ic_launcher"
    />

</FrameLayout>
```

我们指定 TextView 在 FrameLayout 中居左对齐，指定 ImageView 在 FrameLayout 中居右对齐，然后重新运行程序，效果如图 3.24 所示。

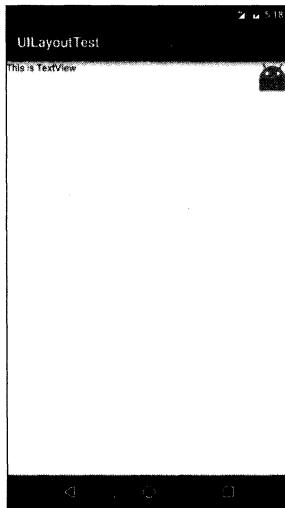


图 3.24 指定 layout_gravity 的效果

总体来讲，FrameLayout 由于定位方式的欠缺，导致它的应用场景也比较少，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 百分比布局

前面介绍的 3 种布局都是从 Android 1.0 版本中就开始支持了，一直沿用到现在，可以说是满足了绝大多数场景的界面设计需求。不过细心的你会发现，只有 LinearLayout 支持使用 layout_weight 属性来实现按比例指定控件大小的功能，其他两种布局都不支持。比如说，如果想用 RelativeLayout 来实现让两个按钮平分布局宽度的效果，则是比较困难的。

为此，Android 引入了一种全新的布局方式来解决此问题——百分比布局。在这种布局中，我们可以不再使用 `wrap_content`、`match_parent` 等方式来指定控件的大小，而是允许直接指定控件在布局中所占的百分比，这样的话就可以轻松实现平分布局甚至是任意比例分割布局的效果了。

由于 `LinearLayout` 本身已经支持按比例指定控件的大小了，因此百分比布局只为 `FrameLayout` 和 `RelativeLayout` 进行了功能扩展，提供了 `PercentFrameLayout` 和 `PercentRelativeLayout` 这两个全新的布局，下面我们就来具体学习一下。

不同于前 3 种布局，百分比布局属于新增布局，那么怎么才能做到让新增布局在所有 Android 版本上都能使用呢？为此，Android 团队将百分比布局定义在了 `support` 库当中，我们只需要在项目的 `build.gradle` 中添加百分比布局库的依赖，就能保证百分比布局在 Android 所有系统版本上的兼容性了。

打开 `app/build.gradle` 文件，在 `dependencies` 闭包中添加如下内容：

```
dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    compile 'com.android.support:appcompat-v7:24.2.1'
    compile 'com.android.support:percent:24.2.1'
    testCompile 'junit:junit:4.12'
}
```

需要注意的是，每当修改了任何 `gradle` 文件时，Android Studio 都会弹出一个如图 3.25 所示的提示。



Gradle files have changed since last project sync. A project sync may be necessary for the IDE to work properly. [Sync Now](#)

图 3.25 `gradle` 文件修改后的提示

这个提示告诉我们，`gradle` 文件自上次同步之后又发生了变化，需要再次同步才能使项目正常工作。这里只需要点击 `Sync Now` 就可以了，然后 `gradle` 会开始进行同步，把我们新添加的百分比布局库引入到项目当中。

接下来修改 `activity_main.xml` 中的代码，如下所示：

```
<android.support.percent.PercentFrameLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <Button
        android:id="@+id/button1"
        android:text="Button 1"
        android:layout_gravity="left|top"
        app:layout_widthPercent="50%"
        app:layout_heightPercent="50%">
```

```

/>

<Button
    android:id="@+id/button2"
    android:text="Button 2"
    android:layout_gravity="right|top"
    app:layout_widthPercent="50%"
    app:layout_heightPercent="50%"
/>

<Button
    android:id="@+id/button3"
    android:text="Button 3"
    android:layout_gravity="left|bottom"
    app:layout_widthPercent="50%"
    app:layout_heightPercent="50%"
/>

<Button
    android:id="@+id/button4"
    android:text="Button 4"
    android:layout_gravity="right|bottom"
    app:layout_widthPercent="50%"
    app:layout_heightPercent="50%"
/>

</android.support.percent.PercentFrameLayout>

```

最外层我们使用了 PercentFrameLayout，由于百分比布局并不是内置在系统 SDK 当中的，所以需要把完整的包路径写出来。然后还必须定义一个 app 的命名空间，这样才能使用百分比布局的自定义属性。

在 PercentFrameLayout 中我们定义了 4 个按钮，使用 app:layout_widthPercent 属性将各按钮的宽度指定为布局的 50%，使用 app:layout_heightPercent 属性将各按钮的高度指定为布局的 50%。这里之所以能使用 app 前缀的属性就是因为刚才定义了 app 的命名空间，当然我们一直能使用 android 前缀的属性也是同样的道理。

不过 PercentFrameLayout 还是会继承 FrameLayout 的特性，即所有的控件默认都是摆放在布局的左上角。那么为了让这 4 个按钮不会重叠，这里还是借助了 layout_gravity 来分别将这 4 个按钮放置在布局的左上、右上、左下、右下 4 个位置。

现在我们已经可以重新运行程序了，不过如果你使用的是老版本的 Android Studio，可能会在 activity_main.xml 中看到一些如图 3.26 所示的错误提示。

'layout_height' attribute should be defined more... (Ctrl+F1)
 'layout_width' attribute should be defined more... (Ctrl+F1)

图 3.26 activity_main.xml 中错误提示

这是因为老版本的Android Studio中内置了布局的检查机制，认为每一个控件都应该通过`android:layout_width`和`android:layout_height`属性指定宽高才是合法的。而其实我们是通过`app:layout_widthPercent`和`app:layout_heightPercent`属性来指定宽高的，所以Android Studio没检测到。不过这个错误提示并不影响程序运行，我们直接忽视就可以了。当然最新的Android Studio 2.2版本中已经修复了这个问题，因此你可能并不会看到上述的错误提示。

现在重新运行程序，效果如图3.27所示。

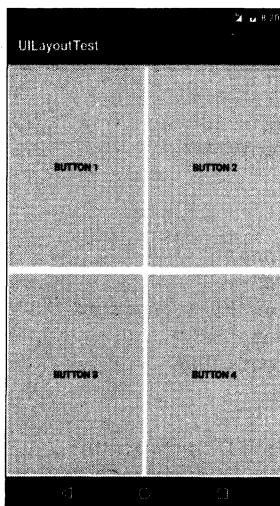


图3.27 PercentFrameLayout运行效果

可以看到，每一个按钮的宽和高都占据了布局的50%，这样我们就轻松实现了4个按钮平分屏幕的效果。

PercentFrameLayout的用法就介绍到这里，另外一个PercentRelativeLayout的用法也是非常相似的，它继承了RelativeLayout中的所有属性，并且可以使用`app:layout_widthPercent`和`app:layout_heightPercent`来按百分比指定控件的宽高，相信聪明的你一定可以举一反三了。

这样我们就把最常用的几种布局都讲解完了，其实Android中还有AbsoluteLayout、TableLayout等布局，不过由于使用得实在是太少了，就不在本书中进行讲解了。

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中的一些常用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，现在是时候来看一下了，如图3.28所示。

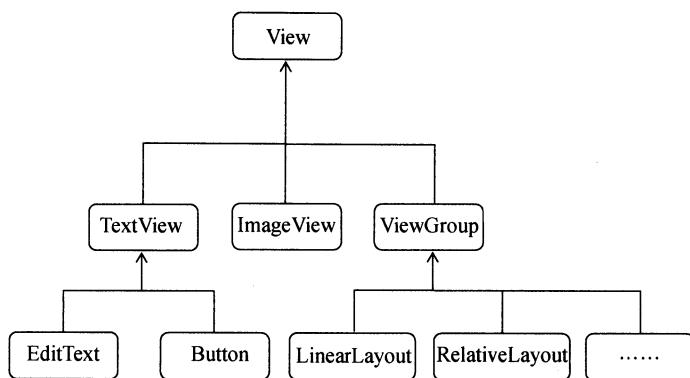


图 3.28 常用控件和布局的继承结构

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 中最基本的一种 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，当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 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，但这里我们决定先不使用它，而是创建一个自定义的标题栏。

经过前面两节的学习，相信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="5dp"
    android:background="@drawable/back_bg"
    android:text="Back"
    android:textColor="#fff" />

    <TextView
        android:id="@+id/title_text"
        android:layout_width="0dp"
        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="5dp"
        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。上面代码中的大多数属性都是你已经见过的，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 3 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 AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        ActionBar actionBar = getSupportActionBar();
        if (actionBar != null) {
            actionBar.hide();
        }
    }
}
```

这里我们调用了 `getSupportActionBar()` 方法来获得 `ActionBar` 的实例，然后再调用 `ActionBar` 的 `hide()` 方法将标题栏隐藏起来。关于 `ActionBar` 的更多用法我们将会在第 12 章中讲解，现在你只需要知道可以通过这种写法来隐藏标题栏就足够了。现在运行一下程序，效果如图 3.29 所示。

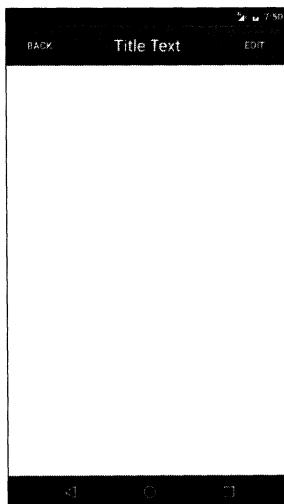


图 3.29 引入标题栏布局的效果

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，无疑会增加很多重复代码，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" />

</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。

下面我们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 3.30 所示。

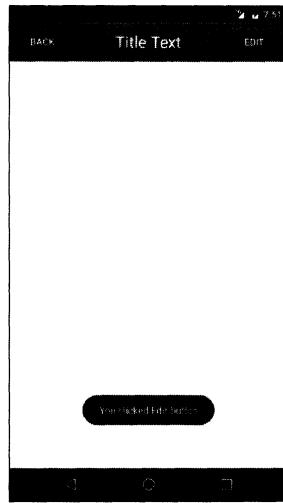


图 3.30 点击编辑按钮的效果

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

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 QQ 聊天记录，翻阅微博最新消息，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 Android Studio 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" />

</LinearLayout>
```

在布局中加入 ListView 控件还算非常简单，先为 ListView 指定一个 id，然后将宽度和高度都设置为 `match_parent`，这样 ListView 也就占满了整个布局的空间。

接下来修改 MainActivity 中的代码，如下所示：

```
public class MainActivity extends AppCompatActivity {

    private String[] data = { "Apple", "Banana", "Orange", "Watermelon",
        "Pear", "Grape", "Pineapple", "Strawberry", "Cherry", "Mango",
        "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而定。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。`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 3.31 所示。可以通过滚动的方式来查看屏幕外的数据。

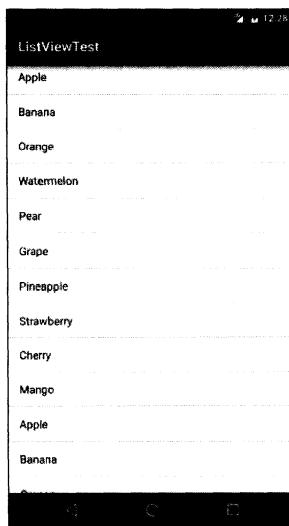


图 3.31 ListView 运行效果

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="wrap_content">

    <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_vertical"
        android:layout_marginLeft="10dp" />

</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，并让 TextView 在垂直方向上居中显示。

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, parent,
        false);
    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。

这里 `LayoutInflater` 的 `inflate()`方法接收 3 个参数，前两个参数我们已经知道是什么意思了，第三个参数指定成 `false`，表示只让我们在父布局中声明的 `layout` 属性生效，但不为这个 `View` 添加父布局，因为一旦 `View` 有了父布局之后，它就不能再添加到 `ListView` 中了。如果你现在还不能理解这段话的含义也没关系，只需要知道这是 `ListView` 中的标准写法就可以了，当你以后对 `View` 理解得更加深刻的时候，再来读这段话就没有问题了。

我们继续往下看，接下来调用 `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 AppCompatActivity {

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

    @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() {
        for (int i = 0; i < 2; i++) {
            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。另外我们使用了一个 `for` 循环将所有的水果数据添加了两遍，这是因为如果只添加一遍的话，数据量还不足以充满整个屏幕。接着在 `onCreate()` 方法中创建了 `FruitAdapter` 对象，并将 `FruitAdapter` 作为适配器传递给 `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 3.32 所示。

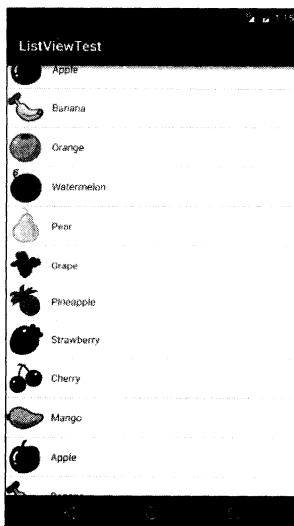


图 3.32 定制界面的 ListView 运行效果

虽然目前我们定制的界面还很简单，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。

仔细观察会发现，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, parent,
                false);
        } 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 null，则使用 LayoutInflater 去加载布局，如果不为 null 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, parent,
            false);
        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 null 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 null 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 AppCompatActivity {

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

```

```
@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 AdapterView.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 `setOnItemClickListener()` 方法为 `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，并点击一下橘子，效果如图 3.33 所示。



图 3.33 点击 ListView 的效果

3.6 更强大的滚动控件——RecyclerView

ListView由于其强大的功能，在过去的Android开发当中可以说是贡献卓越，直到今天仍然还有不计其数的程序在继续使用着ListView。不过ListView并不是完全没有缺点的，比如说如果我们不使用一些技巧来提升它的运行效率，那么ListView的性能就会非常差。还有，ListView的扩展性也不够好，它只能实现数据纵向滚动的效果，如果我们想实现横向滚动的话，ListView是做不到的。

为此，Android提供了一个更强大的滚动控件——RecyclerView。它可以说是一个增强版的ListView，不仅可以轻松实现和ListView同样的效果，还优化了ListView中存在的各种不足之处。目前Android官方更加推荐使用RecyclerView，未来也会有更多的程序逐渐从ListView转向RecyclerView，那么本节我们就来详细讲解一下RecyclerView的用法。

首先新建一个RecyclerViewTest项目，并让Android Studio自动帮我们创建好活动。

3.6.1 RecyclerView的基本用法

和百分比布局类似，RecyclerView也属于新增的控件，为了让RecyclerView在所有Android版本上都能使用，Android团队采取了同样的方式，将RecyclerView定义在了support库当中。因此，想要使用RecyclerView这个控件，首先需要在项目的build.gradle中添加相应的依赖库才行。

打开app/build.gradle文件，在dependencies闭包中添加如下内容：

```
dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    compile 'com.android.support:appcompat-v7:24.2.1'
    compile 'com.android.support:recyclerview-v7:24.2.1'
    testCompile 'junit:junit:4.12'
}
```

添加完之后记得要点击一下Sync Now来进行同步。然后修改activity_main.xml中的代码，如下所示：

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

    <android.support.v7.widget.RecyclerView
        android:id="@+id/recycler_view"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />

</LinearLayout>
```

在布局中加入RecyclerView控件也是非常简单的，先为RecyclerView指定一个id，然后将宽度和高度都设置为match_parent，这样RecyclerView也就占满了整个布局的空间。需要注意的是，由于RecyclerView并不是内置在系统SDK当中的，所以需要把完整的包路径写出来。

这里我们想要使用 RecyclerView 来实现和 ListView 相同的效果，因此就需要准备一份同样的水果图片。简单起见，我们就直接从 ListViewTest 项目中把图片复制过来就可以了，另外顺便将 Fruit 类和 fruit_item.xml 也复制过来，省得将同样的代码再写一遍。

接下来需要为 RecyclerView 准备一个适配器，新建 FruitAdapter 类，让这个适配器继承自 RecyclerView.Adapter，并将泛型指定为 FruitAdapter.ViewHolder。其中，ViewHolder 是我们在 FruitAdapter 中定义的一个内部类，代码如下所示：

```
public class FruitAdapter extends RecyclerView.Adapter<FruitAdapter.ViewHolder> {

    private List<Fruit> mFruitList;

    static class ViewHolder extends RecyclerView.ViewHolder {
        ImageView fruitImage;
        TextView fruitName;

        public ViewHolder(View view) {
            super(view);
            fruitImage = (ImageView) view.findViewById(R.id.fruit_image);
            fruitName = (TextView) view.findViewById(R.id.fruit_name);
        }
    }

    public FruitAdapter(List<Fruit> fruitList) {
        mFruitList = fruitList;
    }

    @Override
    public ViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {
        View view = LayoutInflater.from(parent.getContext())
            .inflate(R.layout.fruit_item, parent, false);
        ViewHolder holder = new ViewHolder(view);
        return holder;
    }

    @Override
    public void onBindViewHolder(ViewHolder holder, int position) {
        Fruit fruit = mFruitList.get(position);
        holder.fruitImage.setImageResource(fruit.getImageId());
        holder.fruitName.setText(fruit.getName());
    }

    @Override
    public int getItemCount() {
        return mFruitList.size();
    }
}
```

虽然这段代码看上去好像有点长，但其实它比 ListView 的适配器要更容易理解。这里我们首先定义了一个内部类 ViewHolder，ViewHolder 要继承自 RecyclerView.ViewHolder。然后 ViewHolder 的构造函数中要传入一个 View 参数，这个参数通常就是 RecyclerView 子项的最外

层布局，那么我们就可以通过 `findViewById()` 方法来获取到布局中的 `ImageView` 和 `TextView` 的实例了。

接着往下看，`FruitAdapter` 中也有一个构造函数，这个方法用于把要展示的数据源传进来，并赋值给一个全局变量 `mFruitList`，我们后续的操作都将在这个数据源的基础上进行。

继续往下看，由于 `FruitAdapter` 是继承自 `RecyclerView.Adapter` 的，那么就必须重写 `onCreateViewHolder()`、`onBindViewHolder()` 和 `getItemCount()` 这 3 个方法。`onCreateViewHolder()` 方法是用于创建 `ViewHolder` 实例的，我们在这个方法中将 `fruit_item` 布局加载进来，然后创建一个 `ViewHolder` 实例，并把加载出来的布局传入到构造函数当中，最后将 `ViewHolder` 的实例返回。`onBindViewHolder()` 方法是用于对 `RecyclerView` 子项的数据进行赋值的，会在每个子项被滚动到屏幕内的时候执行，这里我们通过 `position` 参数得到当前项的 `Fruit` 实例，然后再将数据设置到 `ViewHolder` 的 `ImageView` 和 `TextView` 当中即可。`getCount()` 方法就非常简单了，它用于告诉 `RecyclerView` 一共有多少子项，直接返回数据源的长度就可以了。

适配器准备好了之后，我们就可以开始使用 `RecyclerView` 了，修改 `MainActivity` 中的代码，如下所示：

```
public class MainActivity extends AppCompatActivity {

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

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        initFruits(); // 初始化水果数据
        RecyclerView recyclerView = (RecyclerView) findViewById(R.id.recycler_view);
        LinearLayoutManager layoutManager = new LinearLayoutManager(this);
        recyclerView.setLayoutManager(layoutManager);
        FruitAdapter adapter = new FruitAdapter(fruitList);
        recyclerView.setAdapter(adapter);
    }

    private void initFruits() {
        for (int i = 0; i < 2; i++) {
            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，这里使用了一个同样的 `initFruits()`方法，用于初始化所有的水果数据。接着在 `onCreate()`方法中我们先获取到 `RecyclerView` 的实例，然后创建了一个 `LinearLayoutManager` 对象，并将它设置到 `RecyclerView` 当中。`LayoutManager` 用于指定 `RecyclerView` 的布局方式，这里使用的 `LinearLayoutManager` 是线性布局的意思，可以实现和 `ListView` 类似的效果。接下来我们创建了 `FruitAdapter` 的实例，并将水果数据传入到 `FruitAdapter` 的构造函数中，最后调用 `RecyclerView` 的 `setAdapter()` 方法来完成适配器设置，这样 `RecyclerView` 和数据之间的关联就建立完成了。

现在可以运行一下程序了，效果如图 3.34 所示。

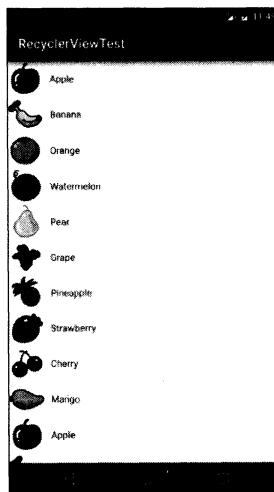


图 3.34 RecyclerView 运行效果

可以看到，我们使用 RecyclerView 实现了和 ListView 几乎一模一样的效果，虽说在代码量方面并没有明显地减少，但是逻辑变得更加清晰了。当然这只是 RecyclerView 的基本用法而已，接下来我们就看一看 RecyclerView 还能实现哪些 ListView 实现不了的效果。

3.6.2 实现横向滚动和瀑布流布局

我们已经知道，ListView 的扩展性并不好，它只能实现纵向滚动的效果，如果想进行横向滚动

动的话，ListView 就做不到了。那么 RecyclerView 就能做得到吗？当然可以，不仅能做到，还非常简单，那么接下来我们就尝试实现一下横向滚动的效果。

首先要对 fruit_item 布局进行修改，因为目前这个布局里面的元素是水平排列的，适用于纵向滚动的场景，而如果我们要实现横向滚动的话，应该把 fruit_item 里的元素改成垂直排列才比较合理。修改 fruit_item.xml 中的代码，如下所示：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="100dp"
    android:layout_height="wrap_content" >

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

    <TextView
        android:id="@+id/fruit_name"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center_horizontal"
        android:layout_marginTop="10dp" />

</LinearLayout>
```

可以看到，我们将 LinearLayout 改成垂直方向排列，并把宽度设为 100dp。这里将宽度指定为固定值是因为每种水果的文字长度不一致，如果用 wrap_content 的话，RecyclerView 的子项就会有长有短，非常不美观；而如果用 match_parent 的话，就会导致宽度过长，一个子项占满整个屏幕。

然后我们将 ImageView 和 TextView 都设置成了在布局中水平居中，并且使用 layout_marginTop 属性让文字和图片之间保持一些距离。

接下来修改 MainActivity 中的代码，如下所示：

```
public class MainActivity extends AppCompatActivity {

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

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        initFruits();
        RecyclerView recyclerView = (RecyclerView) findViewById(R.id.recycler_view);
        LinearLayoutManager layoutManager = new LinearLayoutManager(this);
        layoutManager.setOrientation(LinearLayoutManager.HORIZONTAL);
        recyclerView.setLayoutManager(layoutManager);
```

```

        FruitAdapter adapter = new FruitAdapter(fruitList);
        recyclerView.setAdapter(adapter);
    }
    ...
}

```

MainActivity 中只加入了一行代码，调用 LinearLayoutManager 的 setOrientation()方法来设置布局的排列方向，默认是纵向排列的，我们传入 LinearLayoutManager.HORIZONTAL 表示让布局横向排列，这样 RecyclerView 就可以横向滚动了。

重新运行一下程序，效果如图 3.35 所示。

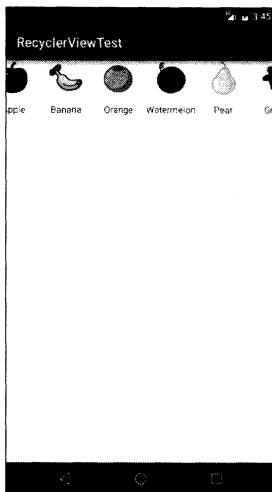


图 3.35 横向 RecyclerView 效果

你可以用手指在水平方向上滑动来查看屏幕外的数据。

为什么 ListView 很难或者根本无法实现的效果在 RecyclerView 上这么轻松就能实现了呢？这主要得益于 RecyclerView 出色的设计。ListView 的布局排列是由自身去管理的，而 RecyclerView 则将这个工作交给了 LayoutManager，LayoutManager 中制定了一套可扩展的布局排列接口，子类只要按照接口的规范来实现，就能定制出各种不同排列方式的布局了。

除了 LinearLayoutManager 之外，RecyclerView 还给我们提供了 GridLayoutManager 和 StaggeredGridLayoutManager 这两种内置的布局排列方式。GridLayoutManager 可以用于实现网格布局，StaggeredGridLayoutManager 可以用于实现瀑布流布局。这里我们来实现一下效果更加炫酷的瀑布流布局，网格布局就作为课后习题，交给你自己来研究了。

首先还是来修改一下 fruit_item.xml 中的代码，如下所示：

```

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

```

```

    android:layout_height="wrap_content"
    android:layout_margin="5dp" >

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

    <TextView
        android:id="@+id/fruit_name"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="left"
        android:layout_marginTop="10dp" />

</LinearLayout>

```

这里做了几处小的调整，首先将 LinearLayout 的宽度由 100dp 改成了 match_parent，因为瀑布流布局的宽度应该是根据布局的列数来自动适配的，而不是一个固定值。另外我们使用了 layout_margin 属性来让子项之间互留一点间距，这样就不至于所有子项都紧贴在一起。还有就是将 TextView 的对齐属性改成了居左对齐，因为待会我们会将文字的长度变长，如果还是居中显示就会感觉怪怪的。

接着修改 MainActivity 中的代码，如下所示：

```

public class MainActivity extends AppCompatActivity {

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

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        initFruits();
        RecyclerView recyclerView = (RecyclerView) findViewById(R.id.recycler_view);
        StaggeredGridLayoutManager layoutManager = new
        StaggeredGridLayoutManager(3, StaggeredGridLayoutManager.VERTICAL);
        recyclerView.setLayoutManager(layoutManager);
        FruitAdapter adapter = new FruitAdapter(fruitList);
        recyclerView.setAdapter(adapter);
    }

    private void initFruits() {
        for (int i = 0; i < 2; i++) {
            Fruit apple = new Fruit(
                getRandomLengthName("Apple"), R.drawable.apple_pic);
            fruitList.add(apple);
            Fruit banana = new Fruit(
                getRandomLengthName("Banana"), R.drawable.banana_pic);
            fruitList.add(banana);
            Fruit orange = new Fruit(

```

```

        getRandomLengthName("Orange"), R.drawable.orange_pic);
fruitList.add(orange);
Fruit watermelon = new Fruit(
        getRandomLengthName("Watermelon"), R.drawable.watermelon_pic);
fruitList.add(watermelon);
Fruit pear = new Fruit(
        getRandomLengthName("Pear"), R.drawable.pear_pic);
fruitList.add(pear);
Fruit grape = new Fruit(
        getRandomLengthName("Grape"), R.drawable.grape_pic);
fruitList.add(grape);
Fruit pineapple = new Fruit(
        getRandomLengthName("Pineapple"), R.drawable.pineapple_pic);
fruitList.add(pineapple);
Fruit strawberry = new Fruit(
        getRandomLengthName("Strawberry"), R.drawable.strawberry_pic);
fruitList.add(strawberry);
Fruit cherry = new Fruit(
        getRandomLengthName("Cherry"), R.drawable.cherry_pic);
fruitList.add(cherry);
Fruit mango = new Fruit(
        getRandomLengthName("Mango"), R.drawable.mango_pic);
fruitList.add(mango);
}
}

private String getRandomLengthName(String name) {
    Random random = new Random();
    int length = random.nextInt(20) + 1;
    StringBuilder builder = new StringBuilder();
    for (int i = 0; i < length; i++) {
        builder.append(name);
    }
    return builder.toString();
}
}

```

首先，在`onCreate()`方法中，我们创建了一个`StaggeredGridLayoutManager`的实例。`StaggeredGridLayoutManager`的构造函数接收两个参数，第一个参数用于指定布局的列数，传入3表示会把布局分为3列；第二个参数用于指定布局的排列方向，传入`StaggeredLayoutManager.VERTICAL`表示会让布局纵向排列，最后再把创建好的实例设置到`RecyclerView`当中就可以了，就是这么简单！

没错，仅仅修改了一行代码，我们就已经成功实现瀑布流布局的效果了。不过由于瀑布流布局需要各个子项的高度不一致才能看出明显的效果，为此我又使用了一个小技巧。这里我们把目光聚焦在`getRandomLengthName()`这个方法上，这个方法使用了`Random`对象来创造一个1到20之间的随机数，然后将参数中传入的字符串重复随机遍。在`initFruits()`方法中，每个水果的名字都改成调用`getRandomLengthName()`这个方法来生成，这样就能保证各水果名字的长短

差距都比较大，子项的高度也就各不相同了。

现在重新运行一下程序，效果如图 3.36 所示。

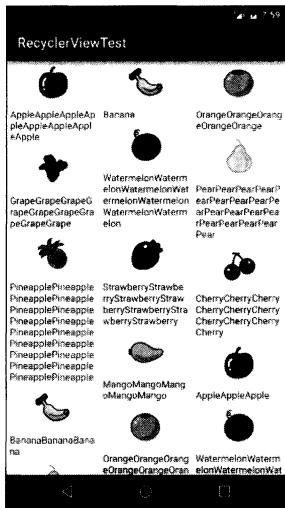


图 3.36 瀑布流布局效果

当然由于水果名字的长度每次都是随机生成的，你运行时的效果肯定和图中还是不一样的。

3.6.3 RecyclerView 的点击事件

和 ListView 一样，RecyclerView 也必须要能影响点击事件才可以，不然的话就没什么实际用途了。不过不同于 ListView 的是，RecyclerView 并没有提供类似于 `setOnItemClickListener()` 这样的注册监听器方法，而是需要我们自己给子项具体的 View 去注册点击事件，相比于 ListView 来说，实现起来要复杂一些。

那么你可能就有疑问了，为什么 RecyclerView 在各方面的设计都要优于 ListView，偏偏在点击事件上却没有处理得非常好呢？其实不是这样的，ListView 在点击事件上的处理并不人性化，`setOnItemClickListener()` 方法注册的是子项的点击事件，但如果我想点击的是子项里具体的某一个按钮呢？虽然 ListView 也是能做到的，但是实现起来就相对比较麻烦了。为此，RecyclerView 干脆直接摒弃了子项点击事件的监听器，所有的点击事件都由具体的 View 去注册，就再没有这个困扰了。

下面我们来具体学习一下如何在 RecyclerView 中注册点击事件，修改 `FruitAdapter` 中的代码，如下所示：

```
public class FruitAdapter extends RecyclerView.Adapter<FruitAdapter.ViewHolder> {
    private List<Fruit> mFruitList;
```

```

static class ViewHolder extends RecyclerView.ViewHolder {
    View fruitView;
    ImageView fruitImage;
    TextView fruitName;

    public ViewHolder(View view) {
        super(view);
        fruitView = view;
        fruitImage = (ImageView) view.findViewById(R.id.fruit_image);
        fruitName = (TextView) view.findViewById(R.id.fruit_name);
    }
}

public FruitAdapter(List<Fruit> fruitList) {
    mFruitList = fruitList;
}

@Override
public ViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {
    View view = LayoutInflater.from(parent.getContext()).inflate(R.layout.
        fruit_item, parent, false);
    final ViewHolder holder = new ViewHolder(view);
    holder.fruitView.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            int position = holder.getAdapterPosition();
            Fruit fruit = mFruitList.get(position);
            Toast.makeText(v.getContext(), "you clicked view " + fruit.getName(),
                Toast.LENGTH_SHORT).show();
        }
    });
    holder.fruitImage.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            int position = holder.getAdapterPosition();
            Fruit fruit = mFruitList.get(position);
            Toast.makeText(v.getContext(), "you clicked image " + fruit.getName(),
                Toast.LENGTH_SHORT).show();
        }
    });
    return holder;
}

...
}

```

我们先是修改了 ViewHolder，在 ViewHolder 中添加了 fruitView 变量来保存子项最外层布局的实例，然后在 onCreateViewHolder()方法中注册点击事件就可以了。这里分别为最外层布局和 ImageView 都注册了点击事件，RecyclerView 的强大之处也在这里，它可以轻松实现子项中任意控件或布局的点击事件。我们在两个点击事件中先获取了用户点击的 position，然后通过 position 拿到相应的 Fruit 实例，再使用 Toast 分别弹出两种不同的内容以示区别。

现在重新运行代码，并点击香蕉的图片部分，效果如图 3.37 所示。可以看到，这时触发了 ImageView 的点击事件。

然后再点击菠萝的文字部分，由于 TextView 并没有注册点击事件，因此点击文字这个事件会被子项的最外层布局捕获到，效果如图 3.38 所示。

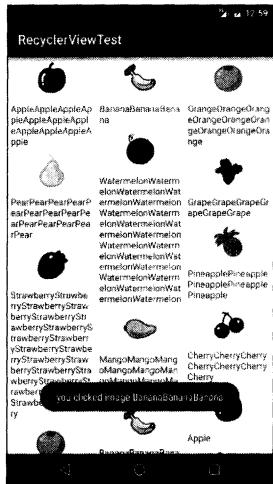


图 3.37 点击香蕉的图片部分

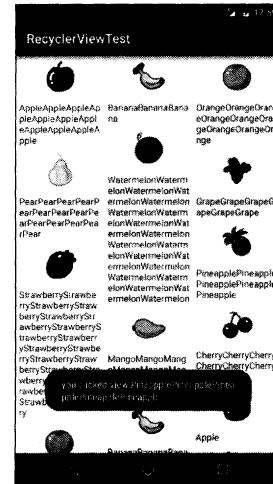


图 3.38 点击菠萝的文字部分

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、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 3.39 所示。



图 3.39 气泡样式图片

我们将这张图片设置为 LinearLayout 的背景图片，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="wrap_content"  
    android:background="@drawable/message_left"  
>  
</LinearLayout>
```

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 3.40 所示。

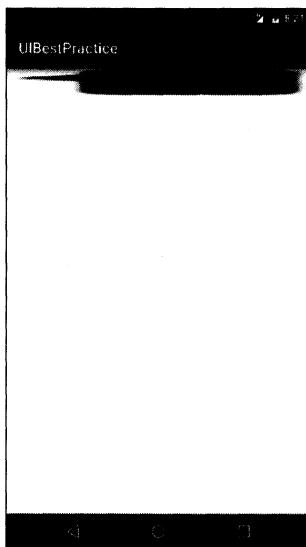


图 3.40 气泡被均匀拉伸的效果

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。不过，要打开这个文件，必须先将 JDK 的 bin 目录配置到环境变量当中才行，比如你使用的是 Android Studio 内置的 jdk，那么要配置的路径就是<Android Studio 安装目录>/jre/bin。如果你还不知道该如何配置环境变量，可以先去参考 6.4.1 小节的内容。

双击打开 `draw9patch.bat` 文件，在导航栏点击 `File→Open 9-patch` 将 `message_left.png` 加载进来，如图 3.41 所示。

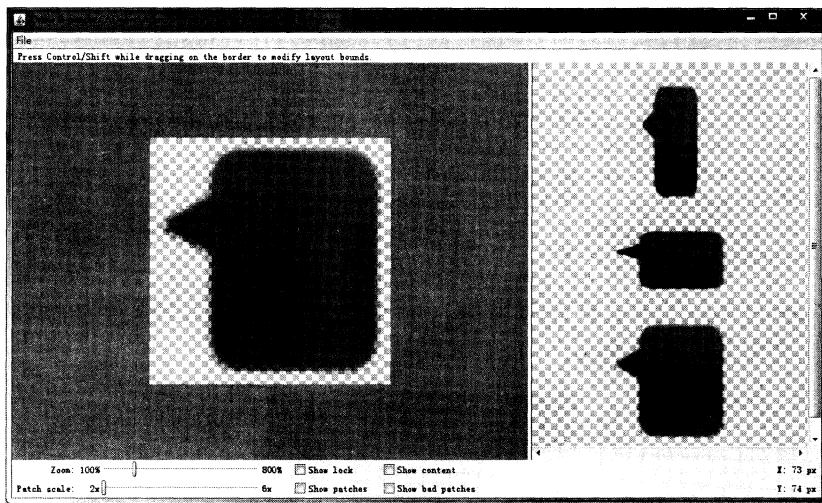


图 3.41 使用 draw9patch 编辑 message_left 图片

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表示内容会被放置的区域。使用鼠标在图片的边缘拖动就可以进行绘制了，按住 Shift 键拖动可以进行擦除。绘制完成后效果如图 3.42 所示。

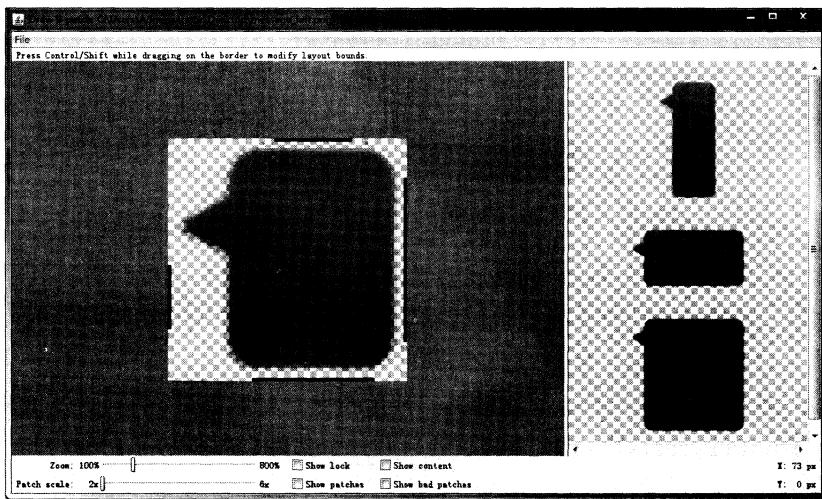


图 3.42 绘制完成后的 message_left 图片

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 3.43 所示。

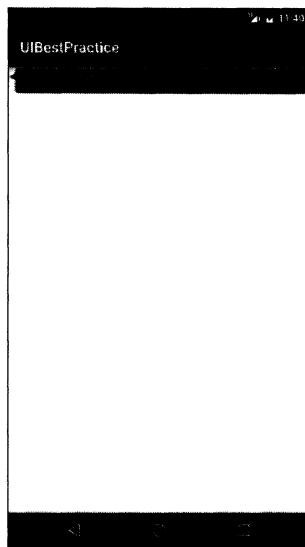


图 3.43 气泡只拉伸绘制区域的效果

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。由于待会我们会用到 RecyclerView，因此首先需要在 app/build.gradle 当中添加依赖库，如下所示：

```
dependencies {  
    compile fileTree(dir: 'libs', include: ['*.jar'])  
    compile 'com.android.support:appcompat-v7:24.2.1'  
    compile 'com.android.support:recyclerview-v7:24.2.1'  
    testCompile 'junit:junit:4.12'  
}
```

接下来开始编写主界面，修改 activity_main.xml 中的代码，如下所示：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"  
    android:orientation="vertical"  
    android:layout_width="match_parent"  
    android:layout_height="match_parent"  
    android:background="#d8e0e8" >  
  
<android.support.v7.widget.RecyclerView
```

```
    android:id="@+id/msg_recycler_view"
    android:layout_width="match_parent"
    android:layout_height="0dp"
    android:layout_weight="1" />

<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 something here"
        android:maxLines="2" />

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

</LinearLayout>

</LinearLayout>
```

我们在主界面中放置了一个 RecyclerView 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。这里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 RecyclerView 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:orientation="vertical"  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"  
    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 RecyclerView 的适配器类，新建类 MsgAdapter，代码如下所示：

```
public class MsgAdapter extends RecyclerView.Adapter<MsgAdapter.ViewHolder> {

    private List<Msg> mMsgList;

    static class ViewHolder extends RecyclerView.ViewHolder {
        LinearLayout leftLayout;
        LinearLayout rightLayout;
        TextView leftMsg;
        TextView rightMsg;

        public ViewHolder(View view) {
            super(view);
            leftLayout = (LinearLayout) view.findViewById(R.id.left_layout);
            rightLayout = (LinearLayout) view.findViewById(R.id.right_layout);
            leftMsg = (TextView) view.findViewById(R.id.left_msg);
            rightMsg = (TextView) view.findViewById(R.id.right_msg);
        }
    }

    public MsgAdapter(List<Msg> msgList) {
        mMsgList = msgList;
    }

    @Override
    public ViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {
        View view = LayoutInflater.from(parent.getContext()).inflate
            (R.layout.msg_item, parent, false);
        return new ViewHolder(view);
    }

    @Override
    public void onBindViewHolder(ViewHolder holder, int position) {
        Msg msg = mMsgList.get(position);
        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
            holder.leftLayout.setVisibility(View.VISIBLE);
            holder.rightLayout.setVisibility(View.GONE);
            holder.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
            holder.rightLayout.setVisibility(View.VISIBLE);
            holder.leftLayout.setVisibility(View.GONE);
            holder.rightMsg.setText(msg.getContent());
        }
    }
}
```

```

        }

    }

    @Override
    public int getItemCount() {
        return mMsgList.size();
    }

}

```

以上代码你应该非常熟悉了，和我们学习 RecyclerView 那一节的代码基本是一样的，只不过在 `onBindViewHolder()` 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 RecyclerView 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 AppCompatActivity {

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

    private EditText inputText;

    private Button send;

    private RecyclerView msgRecyclerView;

    private MsgAdapter adapter;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        initMsgs(); // 初始化消息数据
        inputText = (EditText) findViewById(R.id.input_text);
        send = (Button) findViewById(R.id.send);
        msgRecyclerView = (RecyclerView) findViewById(R.id.msg_recycler_view);
        LinearLayoutManager layoutManager = new LinearLayoutManager(this);
        msgRecyclerView.setLayoutManager(layoutManager);
        adapter = new MsgAdapter(msgList);
        msgRecyclerView.setAdapter(adapter);
        send.setOnClickListener(new View.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.notifyItemInserted(msgList.size() - 1); // 当有新消息时，刷新 ListView 中的显示
                    msgRecyclerView.scrollToPosition(msgList.size() - 1); // 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 RecyclerView 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 null 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 `notifyItemInserted()` 方法，用于通知列表有新的数据插入，这样新增的一条消息才能够在 RecyclerView 中显示。接着调用 RecyclerView 的 `scrollToPosition()` 方法将显示的数据定位到最后一条，以保证一定可以看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 3.44 所示。

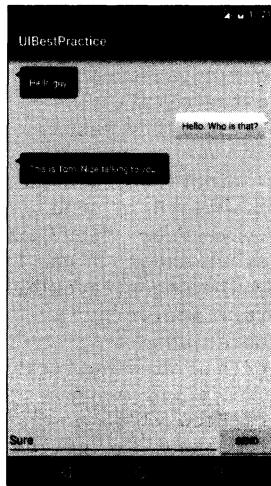


图 3.44 精美的聊天界面

相信这个例子的实战过程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 RecyclerView 的使用，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 非常简单了呢？以后不管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，是没有什么能够难得倒你的。

不过到目前为止，我们还只是学习了 Android 手机方面的开发技巧，下一章将会涉及一些 Android 平板方面的知识点，能够同时兼容手机和平板也是自 Android 4.0 系统开始就支持的特性。适当地放松和休息一段时间后，我们再来继续前行吧！