第 12 章

Android 5.X 新特性详解

2014，Google 携 Android 5.X 重装回归。全新设计的 UI 和更加优化的性能，再一次奠定了 Android 的霸主地位。本章将就 UI 方面 Google 在 Android 5.X 中的改动来向读者做一个简单的汇总，让读者能够深刻领会 Android 5.X 的精髓所在。

学习本章，你将了解以下内容：

- Android 5.X UI 设计初步
- Android 5.X 新增特性分析

12.1 Android 5.X UI 设计初步

Android 5.X 系列开始使用新的设计风格 Material Design 来统一整个 Android 系统的界面设计风格。与之前的设计风格不同，这次的 Material Design 设计将 Android 带到了一个全新的高度，同时 Google 在官网上推出了全新的设计指南——全面地讲解了 Material Design 的整个实现规范与示例如图 12.1 所示。

本次 Material Design 主要强调了以下几个方面的设计：

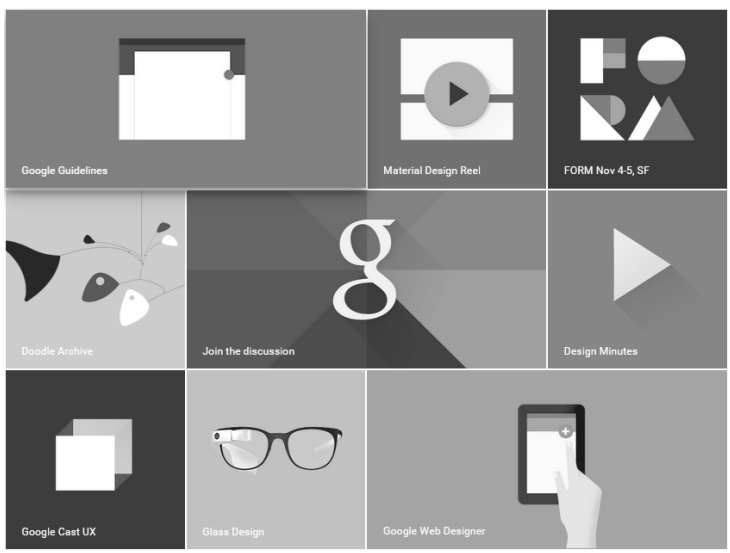

图 12.1 Google Design

12.1.1 材料的形态模拟

材料的形态模拟是 Material Design 中最核心也是改变最大的一个设计，Google 通过模拟自然界纸墨的形态变化、光线与阴影、纸与纸之间的空间层级关系，带来一种真实的空间感，如图 12.2 所示。

12.1.2 更加真实的动画

好的动画效果可以非常有效地指引用户、暗示用户，并给使用者带来非常愉悦的使用体验。Android 5.X 中大量加入了各种新的动画效果，让整个设计风格更加自然、和谐。而各种新的转场动画，能更加有效地指引用户的视觉焦点，不至于因为复杂布局的界面重排而对整体效果

产生影响，让使用者达到一个视觉连贯性。

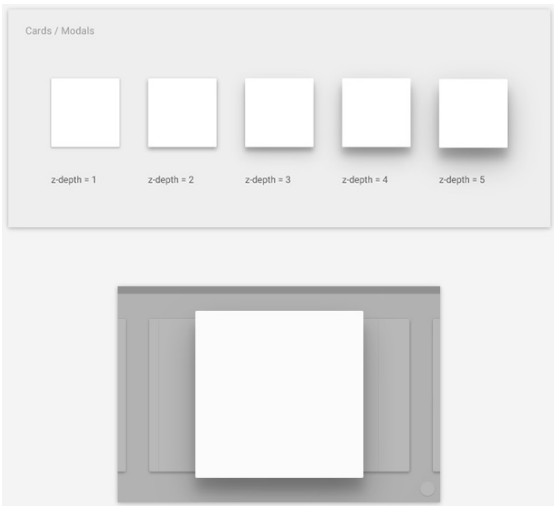

图 12.2　Material Design 空间感

12.1.3　大色块的使用

Material Design 中用大量高饱和度、适中亮度的大色块来突出界面的主次，并一扫 Android 4.X 系列 Holo 主题的沉重感，让界面更加富有时尚感和视觉冲击力，如图 12.3 所示。

图 12.3　大色块的使用

此外，还有很多新的设计风格，比如悬浮按钮、聚焦大图、无框按钮、波纹效果等新特性，这里就不一一列举了。Google 在其 Design 网站上，有整个 Material Design 的设计 Spec，网站如下所示。

http://www.google.com/design/#resources

希望了解更多 Material Design 设计风格的读者可以去这个网站上获得更多的关于 Material Design 的介绍和使用技巧。

12.2 Material Design 主题

首先来看看如何使用 Material Design 的主题。

Material Design 现在有三种默认的主题可以设置，显示效果如图 12.4 所示。

```
@android:style/Theme.Material (dark version)
@android:style/Theme.Material.Light (light version)
@android:style/Theme.Material.Light.DarkActionBar
```

同时，Android 5.X 提出了 Color Palette 的概念，如图 12.5 所示，让开发者可以自己设定系统区域的颜色，使整个 App 的颜色风格和系统的颜色风格保持统一。

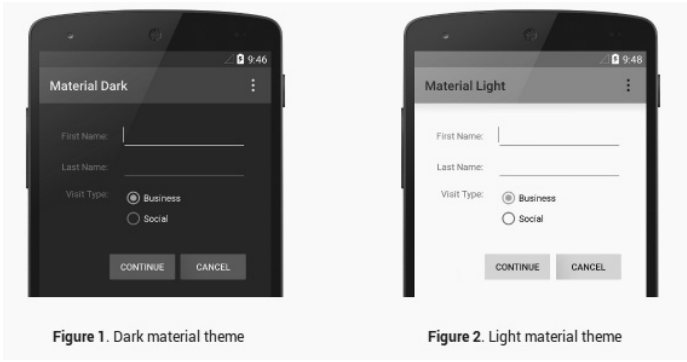

图 12.4 Material Design 主题

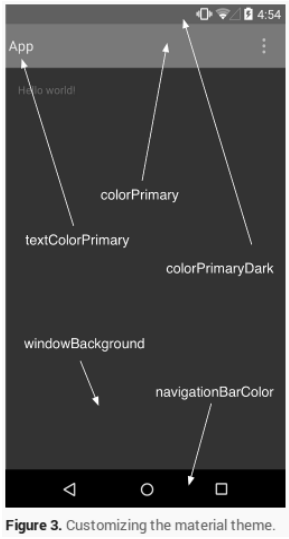

图 12.5 Material Design Color Palette

通过如下所示代码，可以通过使用自定义 Style 的方式来创建自己的 Color Palette 颜色主题，从而实现不同的颜色风格，显示效果如图 12.6 所示。

```
<resources>
    <!-- inherit from the material theme -->
    <style name="AppTheme" parent="android:Theme.Material">
        <!-- Main theme colors -->
        <!--      your app branding color for the app bar -->
        <item name="android:colorPrimary">#BEBEBE</item>

        <!--      darker variant for the status bar and contextual app bars -->
```

```
        <item name="android:colorPrimaryDark">#FF5AEBFF</item>
        <!--      theme UI controls like checkboxes and text fields -->
        <item name="android:navigationBarColor">#FFFF4130</item>
    </style>
</resources>
```

图 12.6　Color Palette 颜色主题

12.3　Palette

在 Android 的版本发展中，UI 越来越成为 Google 的发展重心。这次的 Android 5.X 创新地使用 Palette 来提取颜色，从而让主题能够动态适应当前页面的色调，做到整个 App 颜色基调和谐统一。

Android 内置了几种提取色调的种类，如下所示。

- Vibrant(充满活力的)
- Vibrant dark(充满活力的黑)
- Vibrant light(充满活力的亮)
- Muted(柔和的)
- Muted dark(柔和的黑)
- Muted light(柔和的亮)

使用 Palette 的 API，能够让我们从 Bitmap 中获取对应的色调，修改当前的主题色调。

使用 Palette 首先需要在 Android Studio 中引用相关的依赖，在项目列表上点击 F4，然后在 Module Setting 的 Dependencies 选项卡中添加 com.android.support:palette-v7:21.0.2 引用，重新 Sync 项目即可。可以通过传递一个 Bitmap 对象给 Palette，并调用它的 Palette.generate()静态方法或者 Palette.generateAsync()方法来创建一个 Palette。接下来，就可以使用 getter 方法来检索相应的色调，这些色调就是我们在上面列表中所列出来的色调。

下面这个例子，演示了如何通过加载的图片的柔和色调来改变状态栏和 Actionbar 的色调，代码如下所示。

```
package com.imooc.myapplication;

import android.app.Activity;
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.graphics.drawable.ColorDrawable;
import android.os.Bundle;
import android.support.v7.graphics.Palette;
import android.view.Window;

public class MainActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Bitmap bitmap = BitmapFactory.decodeResource(getResources(),
                R.drawable.test);
        // 创建 Palette 对象
        Palette.generateAsync(bitmap,
                new Palette.PaletteAsyncListener() {
                    @Override
                    public void onGenerated(Palette palette) {
                        // 通过 Palette 来获取对应的色调
                        Palette.Swatch vibrant =
                                palette.getDarkVibrantSwatch();
                        // 将颜色设置给相应的组件
                        getActionBar().setBackgroundDrawable(
                                new ColorDrawable(vibrant.getRgb()));
                        Window window = getWindow();
                        window.setStatusBarColor(vibrant.getRgb());
                    }
                });
    }
}
```

通过以下方法来提取不同色调的颜色。

```
palette.getVibrantSwatch ();
palette.getDarkVibrantSwatch ();
palette.getLightVibrantSwatch ();
palette.getMutedSwatch ();
palette.getDarkMutedSwatch ();
palette.getLightMutedSwatch ();
```

在如上所示代码中使用 getDarkVibrantSwatch()方法提取的色调效果如图 12.7 所示。

图 12.7 设置系统色调

12.4　视图与阴影

Material Design 的一个很重要的特点就是拟物扁平化。如果说 iOS 的扁平化设计太过于超前，让很多人还来不及从拟物转变到扁平，那么 Material Design 则是把 iOS 往回拉了一点。通过展现生活中的材质效果、恰当地使用阴影和光线，配合完美的动画效果，模拟出一个动感十足又美丽大胆的视觉效果。

12.4.1　阴影效果

以往的 Android View 通常具有两个属性——*X* 和 *Y*，而在 Android 5.X 中，Google 为其增加了一个新的属性——*Z*，对应垂直方向上的高度变化。相信通过图 12.8，大家可以很快了解 *Z* 的意义。

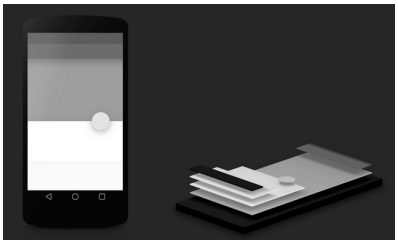

图 12.8　elevation

在 Android 5.X 中，View 的 *Z* 值由两部分组成，elevation 和 translationZ（它们都是 Android 5.X 新引入的属性）。elevation 是静态的成员，translationZ 可以在代码中使用来实现动画效果，它们的关系如下所示。

```
Z = elevation + translationZ
```

通过在 XML 布局中使用如下所示代码来静态设置 View 的视图高度。

```
android:elevation="XXdp"
```

通过下面的代码，演示了不同视图高度所显示效果的不同，XML 代码如下所示，显示效果如图 12.9 所示。

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

    <TextView
        android:layout_width="100dp"

        android:layout_height="100dp"
        android:layout_margin="10dp"
        android:background="@drawable/shape" />

    <TextView
        android:layout_width="100dp"
        android:layout_height="100dp"
        android:layout_margin="10dp"
        android:elevation="2dp"
        android:background="@drawable/shape" />

    <TextView
        android:layout_width="100dp"
        android:layout_height="100dp"
```

```
            android:layout_margin="10dp"
            android:elevation="10dp"
            android:background="@drawable/shape" />
</LinearLayout>
```

图 12.9　视图 elevation

在程序中，同样可以使用如下代码来动态改变视图高度。

```
view.setTranslationZ(XXX);
```

通常也会使用属性动画来为视图高度改变的时候增加一个动画效果，代码如下所示。

```
if (flag) {
    view.animate().translationZ(100);
    flag = false;
} else {
    view.animate().translationZ(0);
    flag = true;
}
```

12.5　Tinting 和 Clipping

Android 5.X 在对图像的操作上有了更多的功能，下面来看看 Android 5.X 的两个对操作图像的新功能——Tinting（着色）和 Clipping（裁剪）。

12.5.1　Tinting（着色）

本例依然拿最熟悉的 Android 小机器人开刀，牺牲一下它的形象来帮助大家理解什么是

Tinting（着色）。

Tinting 的使用非常简单，只要在 XML 中配置好 tint 和 tintMode 就可以了。对于配置的组合效果，只需要大家实际操作一下，就能非常清楚地理解处理效果了，在下面的代码中，设置了几种不同的 tint 和 tintMode 来演示 Tinting 效果，XML 代码如下所示。

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

    android:orientation="vertical"
    tools:context=".MainActivity">

    <ImageView
        android:layout_width="100dp"
        android:layout_height="100dp"
        android:elevation="5dip"
        android:layout_gravity="center"
        android:src="@drawable/ic_launcher" />

    <ImageView
        android:layout_width="100dp"
        android:layout_height="100dp"
        android:elevation="5dip"
        android:layout_gravity="center"
        android:src="@drawable/ic_launcher"
        android:tint="@android:color/holo_blue_bright" />

    <ImageView
        android:layout_width="100dp"
        android:layout_height="100dp"
        android:elevation="5dip"
        android:layout_gravity="center"
        android:src="@drawable/ic_launcher"
        android:tint="@android:color/holo_blue_bright"
        android:tintMode="add" />

    <ImageView
        android:layout_width="100dp"
        android:layout_height="100dp"
        android:elevation="5dip"
        android:layout_gravity="center"
        android:src="@drawable/ic_launcher"
        android:tint="@android:color/holo_blue_bright"
```

```
            android:tintMode="multiply" />

</LinearLayout>
```

效果如图 12.10 所示。

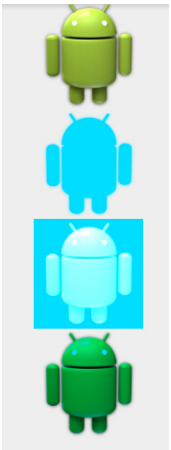

图 12.10　Tint

Tint 通过修改图像的 Alpha 遮罩来修改图像的颜色，从而达到重新着色的目的。这一功能在一些图像处理 App 中使用起来将非常方便。

12.5.2　Clipping（裁剪）

Clipping 可以让我们改变一个视图的外形。要使用 Clipping，首先需要使用 ViewOutlineProvider 来修改 outline，然后再通过 setOutlineProvider 将 outline 作用给视图。

在下面这个例子中，将一个正方形的 TextView 通过 Clipping 裁剪成了一个圆角正方形和一个圆，以此来帮助大家理解 Clipping 的使用思路，XML 代码如下所示。

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

    <TextView
        android:id="@+id/tv_rect"
        android:layout_marginTop="20dp"
        android:layout_width="100dp"
```

```
            android:layout_height="100dp"
            android:elevation="1dip"
            android:layout_gravity="center" />

        <TextView
            android:id="@+id/tv_circle"
            android:layout_width="100dp"
            android:layout_height="100dp"
            android:elevation="1dip"
            android:layout_gravity="center" />

    </LinearLayout>
```

程序代码如下所示。

```
package com.xys.myapplication;

import android.app.Activity;
import android.graphics.Outline;
import android.os.Bundle;
import android.view.View;
import android.view.ViewOutlineProvider;

public class MainActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        View v1 = findViewById(R.id.tv_rect);
        View v2 = findViewById(R.id.tv_circle);
        // 获取 Outline
        ViewOutlineProvider viewOutlineProvider1 =
                new ViewOutlineProvider() {

                    @Override
                    public void getOutline(View view, Outline outline) {
                        // 修改 outline 为特定形状
                        outline.setRoundRect(0, 0,
                                view.getWidth(),
                                view.getHeight(), 30);
                    }
                };
        // 获取 Outline
        ViewOutlineProvider viewOutlineProvider2 =
                new ViewOutlineProvider() {
```

```
                @Override
                public void getOutline(View view, Outline outline) {
                    // 修改 outline 为特定形状
                    outline.setOval(0, 0,
                            view.getWidth(),
                            view.getHeight());
                }
            };
        // 重新设置形状
        v1.setOutlineProvider(viewOutlineProvider1);
        v2.setOutlineProvider(viewOutlineProvider2);
    }
}
```

效果如图 12.11 所示。

图 12.11　Clipping 效果

12.6　列表与卡片

12.6.1　RecyclerView

在 Android5.X 中将使用了很久的 ListView 做了升级，增加了一个使用更方便、效率更高的控件——RecyclerView。RecyclerView 是 support-v7 包中的新组件，是一个强大的滑动组件，与经典的 ListView 相比，它同样拥有 item 回收复用的功能，但是 RecyclerView 可以直接把 ViewHolder 的实现封装起来，用户只要实现自己的 ViewHolder 就可以了，该组件会自动帮你回收复用每一个 item。

要使用 RecyclerView，首先需要在项目中引入 com.android.support:recyclerview- v7:21.0.2 的依赖。在布局中使用 RecyclerView 与使用 ListView 基本类似，同样需要使用一个类似 List item 的布局，在 Material Design 中，通常与 CardView 配合使用，后面我们会详细讲解 CardView 的使用方法。

使用 RecyclerView 的重点与使用 ListView 一样，需要使用一个合适的数据适配器来加载数据，RecyclerView 中需要重写的很多方法都似曾相识，不过 RecyclerView 更加先进的是，它已经封装好了 ViewHolder，只要实现功能就可以了，而使用上仍然是跟在 ListView 中使用 ViewHolder 一样，代码如下所示。

```
package com.xys.myapplication;

import android.support.v7.widget.RecyclerView;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.TextView;
import java.util.List;

public class RecyclerAdapter
        extends RecyclerView.Adapter<RecyclerAdapter.ViewHolder> {

    private List<String> mData;

    public RecyclerAdapter(List<String> data) {
        mData = data;
    }

    public OnItemClickListener itemClickListener;

    public void setOnItemClickListener(
            OnItemClickListener itemClickListener) {
        this.itemClickListener = itemClickListener;
    }

    public interface OnItemClickListener {
        void onItemClick(View view, int position);
    }

    public class ViewHolder extends RecyclerView.ViewHolder

            implements View.OnClickListener {

        public TextView textView;

        public ViewHolder(View itemView) {
            super(itemView);
            textView = (TextView) itemView;
            textView.setOnClickListener(this);
        }
```

```
        // 通过接口回调来实现 RecyclerView 的点击事件
        @Override
        public void onClick(View v) {
            if (itemClickListener != null) {
                itemClickListener.onItemClick(v, getPosition());
            }
        }
    }

    @Override
    public ViewHolder onCreateViewHolder(ViewGroup viewGroup, int i) {
        // 将布局转化为 View 并传递给 RecyclerView 封装好的 ViewHolder
        View v = LayoutInflater.from(viewGroup.getContext()).inflate(
                R.layout.rc_item, viewGroup, false);
        return new ViewHolder(v);
    }

    @Override
    public void onBindViewHolder(ViewHolder viewHolder, int i) {
        // 建立起 ViewHolder 中视图与数据的关联
        viewHolder.textView.setText(mData.get(i) + i);
    }

    @Override

    public int getItemCount() {
        return mData.size();
    }
}
```

上面就是一个非常简单却典型的 RecyclerView，通过 onCreateViewHolder 将 List item 的布局转化为 View，并传递给 RecyclerView 封装好的 ViewHolder，就可以将数据与视图关联起来了。但是有一点要注意的是，Android 并没有给 RecyclerView 增进点击事件，所以我们需要自己使用接口回调机制，创建一个点击事件的接口，代码如下所示。

```
public OnItemClickListener itemClickListener;

public void setOnItemClickListener(OnItemClickListener itemClickListener) {
this.itemClickListener = itemClickListener;
}

public interface OnItemClickListener {
void onItemClick(View view, int position);
}
```

类似 ListView 的 List Item 视图如下所示。

```
<?xml version="1.0" encoding="utf-8"?>
<TextView xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:textSize="40sp"
    android:gravity="center"
    android:layout_width="match_parent"
    android:background="#bebebe"
    android:layout_margin="3dp"
    android:layout_height="match_parent">

</TextView>
```

当然，仅仅是优化性能也是不够的，让开发者能够更加方便地使用也是非常重要的。Google 在 RecyclerView 中定义了 LayoutManager 来帮助开发者更加方便地创建不同的布局，下面的例子就演示了如何创建简单的水平和竖直两种布局方式。当然，你也可以通过自定义 LayoutManager 来创建自己的布局，核心代码如下所示。

```
mRcList.setLayoutManager(new LinearLayoutManager(RecyclerTest.this));
mRcList.setLayoutManager(new GridLayoutManager(RecyclerTest.this, 3));
```

完整代码如下所示。

```
package com.xys.myapplication;

import android.animation.Animator;
import android.animation.AnimatorListenerAdapter;
import android.app.Activity;
import android.os.Bundle;
import android.support.v7.widget.DefaultItemAnimator;
import android.support.v7.widget.GridLayoutManager;
import android.support.v7.widget.LinearLayoutManager;
import android.support.v7.widget.RecyclerView;
import android.view.View;
import android.widget.AdapterView;
import android.widget.Spinner;

import java.util.ArrayList;
import java.util.List;

public class RecyclerTest extends Activity {

    private RecyclerView mRcList;
    private RecyclerAdapter mAdapter;
    private RecyclerView.LayoutManager mLayoutManager;
```

```
private Spinner mSpinner;

private List<String> mData = new ArrayList<String>();

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

    mRcList = (RecyclerView) findViewById(R.id.rc_list);
    mLayoutManager = new LinearLayoutManager(this);
    mRcList.setLayoutManager(mLayoutManager);
    mRcList.setHasFixedSize(true);
    // 设置显示动画
    mRcList.setItemAnimator(new DefaultItemAnimator());

    mSpinner = (Spinner) findViewById(R.id.spinner);
    mSpinner.setOnItemSelectedListener(
            new AdapterView.OnItemSelectedListener() {
                @Override
                public void onItemSelected(AdapterView<?> parent,
                                           View view,
                                           int position,
                                           long id) {
                    if (position == 0) {
                        mRcList.setLayoutManager(
                                // 设置为线性布局
                                new LinearLayoutManager(
                                        RecyclerTest.this));
                    } else if (position == 1) {
                        mRcList.setLayoutManager(
                                // 设置为表格布局
                                new GridLayoutManager(
                                        RecyclerTest.this, 3));
                    } else if (position == 2) {
                    }
                }

                @Override
                public void onNothingSelected(AdapterView<?> parent) {
                }
            });
    // 增加测试数据
    mData.add("Recycler");
```

```
        mData.add("Recycler");
        mData.add("Recycler");
        mAdapter = new RecyclerAdapter(mData);
        mRcList.setAdapter(mAdapter);
        mAdapter.setOnItemClickListener(
            new RecyclerAdapter.OnItemClickListener() {
                @Override
                public void onItemClick(final View view, int position) {
                    // 设置点击动画
                    view.animate()
                    .translationZ(15F).setDuration(300)
                    .setListener(new AnimatorListenerAdapter() {
                        @Override
                        public void onAnimationEnd(Animator animation) {
                            super.onAnimationEnd(animation);
                            view.animate()
                                    .translationZ(1f)
                                    .setDuration(500).start();
                        }
                    }).start();
                }
            });
    }

    public void addRecycler(View view) {
        mData.add("Recycler");
        int position = mData.size();
        if (position > 0) {

            mAdapter.notifyDataSetChanged();
        }
    }

    public void delRecycler(View view) {
        int position = mData.size();
        if (position > 0) {
            mData.remove(position - 1);
            mAdapter.notifyDataSetChanged();
        }
    }
}
```

在程序中，使用 Spinner 来选择线性布局管理器还是表格布局管理器，并给按钮增加了点击动画效果，整个程序运行效果如图 12.12、图 12.13、图 12.14 所示，图 12.12 展示了 RecyclerView 的线性布局，图 12.13、图 12.14 展示了 RecyclerView 的表格布局。

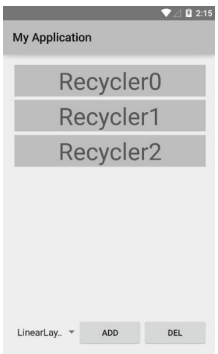

图 12.12　RecyclerView 纵向布局

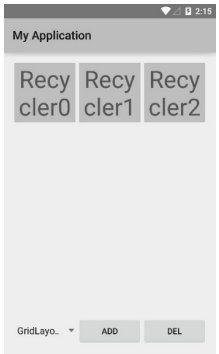

图 12.13　RecyclerView 横向布局

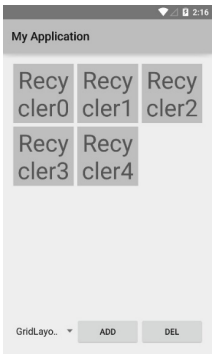

图 12.14　RecyclerView 表格布局

12.6.2　CardView

CardView 曾经开始流行在 Google+上，后来越来越多的 App 也引入了 Card 这样一种布局方式。因此在 Android 5.X 上，Google 索性就提供了 CardView 的控件，方便大家使用这种布局。说到底，CardView 也是一个容器类布局，只是它提供了卡片这样一种形式。开发者可以定义卡片的大小与视图高度，并设置圆角的角度。不过使用 CardView 的方式与 RecyclerView 还是有区别的，首先同样是需要在项目中引入 com.android.support:cardview-v7:21.+的依赖。其次在布局文件中使用CardView的时候需要引入一个新的名字空间——在Android Studio中使用 xmlns:card_view=http://schemas.android. com/apk/res-auto 来添加。这样才可以通过自定义的名字空间来引用它的两个属性。

```
card_view:cardBackgroundColor="@color/cardview_background"
card_view:cardCornerRadius="8dp"
```

这两个属性非常简单，第一个是设置背景颜色，第二个是设置圆角的角度。我们以一个例子来演示一下 CardView 的使用，XML 代码如下所示。

```
<?xml version="1.0" encoding="utf-8"?>

<android.support.v7.widget.CardView xmlns:android="http://schemas.
android.com/apk/res/android"
    xmlns:card_view="http://schemas.android.com/apk/res-auto"
    android:id="@+id/cardview"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:layout_marginTop="40dp"
    card_view:cardBackgroundColor="@color/cardview_initial_background"
    card_view:cardCornerRadius="30dp"
    android:elevation="10dp"
```

```
        android:layout_marginLeft="@dimen/margin_large"
        android:layout_marginRight="@dimen/margin_large">

        <TextView
            android:layout_width="wrap_content"
            android:layout_height="100dp"
            android:gravity="center"
            android:layout_gravity="center"

            android:textSize="30sp"
            android:layout_margin="@dimen/margin_medium"
            android:text="I am a CardView" />
</android.support.v7.widget.CardView>
```

显示效果如图 12.15 所示。

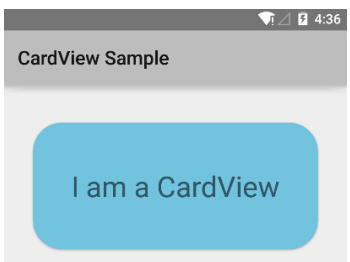

图 12.15　CardView

12.7　Activity 过渡动画

曾经的 Android 在 Activity 进行跳转的时候，只是非常生硬的切换，即使通过 overridePendingtransition(int inId, int outId)这个方法来给 Activity 增加一些切换动画，效果也只是差强人意。而在 Android 5.X 中，Google 对动画效果进行了更深一步的诠释，为 Activity 的转场效果设计了更加丰富的动画效果。

Android 5.X 提供了三种 Transition 类型。

- 进入：一个进入的过渡动画决定 Activity 中的所有的视图怎么进入屏幕。
- 退出：一个退出的过渡动画决定一个 Activity 中的所有视图怎么退出屏幕。
- 共享元素：一个共享元素过渡动画决定两个 Activities 之间的过渡，怎么共享它们的视图。

其中，进入和退出效果包括：

- explode（分解）——从屏幕中间进或出，移动视图
- slide（滑动）——从屏幕边缘进或出，移动视图
- fade（淡出）——通过改变屏幕上视图的不透明度达到添加或者移除视图

共享元素包括：

- changeBounds——改变目标视图的布局边界
- changeClipBounds——裁剪目标视图边界
- changeTransform——改变目标视图的缩放比例和旋转角度
- changeImageTransform——改变目标图片的大小和缩放比例

可以发现，在 Android 5.X 上，动画效果的种类变得更加丰富了。

首先来看看普通的三种 Activity 过渡动画，要使用这些动画非常简单，例如从 ActivityA 跳转到 ActivityB，只需要在 ActivityA 中将基本的 startActivity(intent)方法改为如下代码即可。

```
startActivity(intent, ActivityOptions.makeSceneTransitionAnimation(this). toBundle());
```

而在 ActivityB 中，只需要设置下如下所示代码。

```
getWindow().requestFeature(Window.FEATURE_CONTENT_TRANSITIONS);
```

或者在样式文件中设置如下所示代码。

```
<item name="android:windowContentTransitions">true</item>
```

那么接下来就可以设置进入 ActivityB 的具体的动画效果了，代码如下所示。

```
getWindow().setEnterTransition(new Explode());
getWindow().setEnterTransition(new Slide());
getWindow().setEnterTransition(new Fade());
```

或者通过如下代码来设置离开 ActivityB 的动画效果。

```
getWindow().setExitTransition (new Explode());
getWindow().setExitTransition (new Slide());
getWindow().setExitTransition (new Fade());
```

而对于共享元素的动画效果，可以借用开发者网站上的一张图来帮助大家理解，如图 12.16 所示。

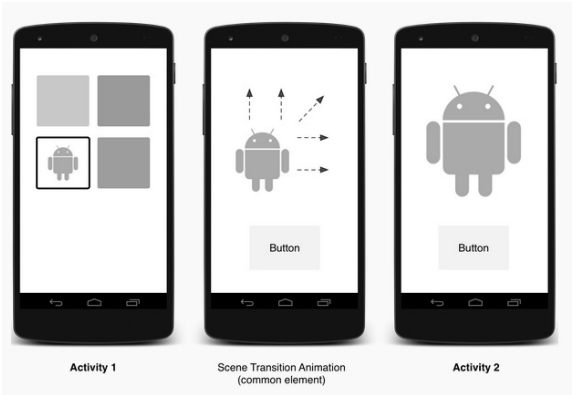

图 12.16　共享元素转场动画

第一张图中的 Android 机器人就是共享元素，即 Activity1 和 Activity2 都拥有的元素，只是在 Activity2 中对 Android 机器人进行了强调，所以视图被放大，在 Activity1 跳转到 Activity2 的时候，其他元素消失，而共享元素——Android 机器人通过动画效果直接显示到 Activity2 中，这个动画效果在 Google IO 大会的 App 上已经有了非常好的展示效果，感兴趣的读者可以下载 Google IO App 来观看其效果。

要想在程序中使用共享元素的动画效果也非常简单，首先需要在 Activity1 的布局文件中设置共享的元素，给它增加相应的属性，代码如下所示。

```
android:transitionName="XXX"
```

同时在 Activity2 的布局文件中，给要实现共享效果的元素也增加相同的属性，代码如下所示。

```
android:transitionName="XXX"
```

这里需要注意的是一定要保证命名相同，这样系统才能找到共享元素。

如果只要一个共享元素，那么在 Activity1 中只需要使用如下代码。

```
startActivity(intent,
        ActivityOptions.makeSceneTransitionAnimation(
                this,
                view,
                "share").toBundle());
```

使用的参数就是在前面普通动画的基础上增加了共享的 View 和前面取的名字。

如果有多个共享的元素，那么可以通过 Pair.create()来创建多个共享元素，代码如下所示。

```
startActivity(intent,
        ActivityOptions.makeSceneTransitionAnimation(
        this,
```

```
            Pair.create(view, "share"),
            Pair.create(fab, "fab")).toBundle());
```

下面通过一个实例来演示 Activity 的过渡动画，Activity1 代码如下所示。

```
package com.yishengxu.myapplication;

import android.app.Activity;
import android.app.ActivityOptions;
import android.content.Intent;
import android.os.Bundle;
import android.util.Pair;
import android.view.View;


public class MainActivity extends Activity {

    private Intent intent;

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

    // 设置不同动画效果
    public void explode(View view) {
        intent = new Intent(this, Transitions.class);
        intent.putExtra("flag", 0);
        startActivity(intent,
                ActivityOptions.makeSceneTransitionAnimation(this)
                        .toBundle());
    }
    // 设置不同动画效果
    public void slide(View view) {
        intent = new Intent(this, Transitions.class);
        intent.putExtra("flag", 1);
        startActivity(intent,
                ActivityOptions.makeSceneTransitionAnimation(this)
                        .toBundle());
    }

    // 设置不同动画效果
    public void fade(View view) {
        intent = new Intent(this, Transitions.class);
        intent.putExtra("flag", 2);
```

```
            startActivity(intent,
                    ActivityOptions.makeSceneTransitionAnimation(this)
                            .toBundle());
        }
        // 设置不同动画效果
        public void share(View view) {
            View fab = findViewById(R.id.fab_button);
            intent = new Intent(this, Transitions.class);
            intent.putExtra("flag", 3);
            // 创建单个共享元素
//              startActivity(intent,
//                      ActivityOptions.makeSceneTransitionAnimation(
//                              this, view, "share").toBundle());
            startActivity(intent,
                    ActivityOptions.makeSceneTransitionAnimation(
                            this,
                            // 创建多个共享元素
                            Pair.create(view, "share"),
                            Pair.create(fab, "fab")).toBundle());
        }
    }
```

XML 代码如下所示。

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:orientation="vertical"
    android:layout_height="match_parent">

    <Button
        android:layout_width="match_parent"
        android:layout_height="100dp"
        android:onClick="explode"
        android:text="explode" />

    <Button
        android:layout_width="match_parent"
        android:layout_height="100dp"
        android:onClick="slide"
        android:text="slide" />

    <Button
        android:layout_width="match_parent"
        android:layout_height="100dp"
        android:onClick="fade"
        android:text="fade" />
```

```
    <Button
        android:layout_width="match_parent"
        android:layout_height="100dp"
        android:transitionName="share"
        android:onClick="share"
        android:text="share" />

    <Button
        android:id="@+id/fab_button"
        android:layout_width="56dp"
        android:transitionName="fab"
        android:layout_height="56dp"
        android:background="@drawable/ripple_round"
        android:elevation="5dp"/>
</LinearLayout>
```

ActivityB 代码如下所示。

```
package com.yishengxu.myapplication;

import android.app.Activity;
import android.os.Bundle;
import android.transition.Explode;
import android.transition.Fade;
import android.transition.Slide;
import android.view.Window;

public class Transitions extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        getWindow().requestFeature(
Window.FEATURE_CONTENT_TRANSITIONS);
        int flag = getIntent().getExtras().getInt("flag");
        // 设置不同的动画效果
        switch (flag) {
            case 0:
                getWindow().setEnterTransition(new Explode());
                break;
            case 1:
                getWindow().setEnterTransition(new Slide());
                break;
            case 2:
                getWindow().setEnterTransition(new Fade());
                getWindow().setExitTransition(new Fade());
                break;
```

```
                case 3:
                    break;
            }
            setContentView(R.layout.activity_transition_to);

        }
    }
```

XML 代码如下所示：

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    tools:context=".MyActivity">

    <View
        android:id="@+id/holder  view"
        android:layout_width="match_parent"
        android:layout_height="300dp"
        android:transitionName="share"
        android:background="?android:colorPrimary" />

    <Button xmlns:android="http://schemas.android.com/apk/res/android"
        android:id="@+id/fab_button"
        android:transitionName="fab"
        android:layout  width="56dp"
        android:layout_height="56dp"
        android:layout_marginRight="@dimen/activity_horizontal_margin"
        android:background="@drawable/ripple_round"
        android:elevation="5dp"
        android:layout_below="@+id/holder_view"
        android:layout_marginTop="-26dp"
        android:layout_alignParentEnd="true" />

    <RelativeLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:paddingTop="10dp"

        android:layout_below="@id/holder_view">

        <Button
            android:layout_width="match_parent"
            android:layout_height="60dp"
            android:id="@+id/button"
            android:layout_below="@+id/button4"
```

```
            android:layout_marginTop="10dp" />

        <Button
            android:layout_width="match_parent"
            android:layout_height="60dp"
            android:layout_marginTop="10dp"
            android:id="@+id/button4"
            android:layout_alignParentStart="true" />

    </RelativeLayout>
</RelativeLayout>
```

程序运行效果如图 12.17、图 12.18 所示。

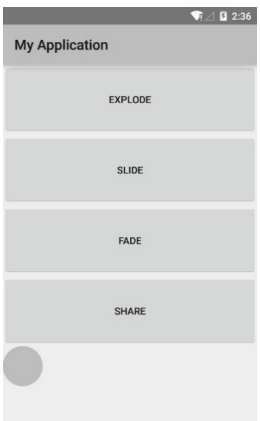

图 12.17　转场动画效果演示（前）

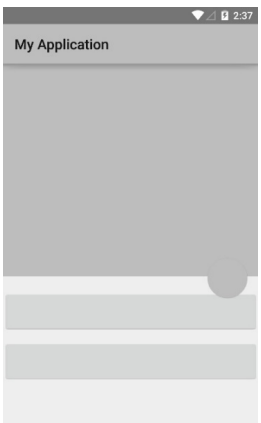

图 12.18 转场动画效果演示（后）

由于本实例涉及很多动态效果，所以很难在书本上让读者体验到实际效果，所以希望读者能实际运行一下本实例，体验 Android 5.X 中的 Activity 转场动画效果。

12.8　Material Design 动画效果

动画已经成了 UI 设计中一个非常重要的组成部分，在 Android 5.X 的 UI 设计 Material Design 中，更是使用了大量的动画效果，同时 Google 也在官方设计文档上增加了对动画的设计指导。

12.8.1　Ripple 效果

在 Android 5.X 中，Material Design 大量使用了 Ripple 效果，即点击后的波纹效果。可以通过如下代码设置波纹的背景。

```
// 波纹有边界
android:background="?android:attr/selectableItemBackground"
// 波纹超出边界
android:background="?android:attr/selectableItemBackgroundBorderless"
```

波纹有边界是指波纹被限制在控件的边界中，而波纹超出边界则是波纹不会限制在控件边界中，会呈圆形发散出去，下面通过一个实例，演示一下 Android 5.X 的波纹效果，XML 代码如下所示。

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@android:color/holo_blue_bright">

    <Button
        android:layout_width="100dp"
        android:layout_height="100dp"
        android:background="?android:attr/selectableItemBackground"
        android:text="有界波纹"
        android:textColor="@android:color/white" />

    <Button
        android:layout_width="100dp"
        android:layout_height="100dp"
        android:background="?android:attr/selectableItemBackground-Borderless"
        android:textColor="@android:color/white"
        android:text="无界波纹" />
</LinearLayout>
```

显示效果如图 12.19、图 12.20 所示。

图 12.19　波纹效果 1

图 12.20　波纹效果 2

同样，你也可以在 XML 文件中直接来创建一个具有 Ripple 效果的 XML 文件，代码如下所示。

```
<ripple
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:color="@android:color/holo_blue_bright">
    <item>
        <shape
```

```
            android:shape="oval">
            <solid android:color="?android:colorAccent" />
        </shape>
    </item>
</ripple>
```

使用方法如下所示，与使用一般的 XML 资源方法相同。

```
<Button
        android:layout_width="100dp"
        android:layout_height="100dp"
        android:background="@drawable/ripple" />
```

显示效果如图 12.21 所示。

图 12.21　波纹效果

12.8.2　Circular Reveal

这个动画效果在 Google IO 大会的演示视频中出现了很多次，具体表现为一个 View 以圆形的形式展开、揭示出来。通过 ViewAnimationUtils.createCircularReveal()方法可以创建一个 RevealAnimator 动画，代码如下所示。

```
public static Animator createCircularReveal(
        View view,
        int centerX,
        int centerY,
        float startRadius,
        float endRadius) {
    return new RevealAnimator(view, centerX, centerY, startRadius, endRadius);
}
```

RevealAnimator 的使用非常简单，主要是设置几个关键的坐标点：

- centerX 动画开始的中心点 X
- centerY 动画开始的中心点 Y
- startRadius 动画开始半径

- startRadius 动画结束半径

通过下面的例子，可以非常直观地感受到这种动画效果，XML 代码如下所示。

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

    <ImageView
        android:id="@+id/oval"
        android:layout_width="100dp"
        android:layout_height="100dp"
        android:background="@drawable/oval" />

    < ImageView
        android:id="@+id/rect"
        android:layout_width="100dp"
        android:layout_height="100dp"
        android:background="@drawable/rect" />
</LinearLayout>
```

程序代码如下所示。

```
package com.xys.myapplication;

import android.animation.Animator;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.view.ViewAnimationUtils;
import android.view.animation.AccelerateDecelerateInterpolator;
import android.view.animation.AccelerateInterpolator;

import com.imooc.myapplication.R;

public class MainActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        final View oval = this.findViewById(R.id.oval);
        oval.setOnClickListener(new View.OnClickListener() {
            @Override
```

```
            public void onClick(View v) {
                Animator animator =
                        ViewAnimationUtils.createCircularReveal(
                        oval,
                        oval.getWidth() / 2,
                        oval.getHeight() / 2,
                        oval.getWidth(),
                        0);
                animator.setInterpolator(
                        new AccelerateDecelerateInterpolator());
                animator.setDuration(2000);
                animator.start();
            }
        });

        final View rect = this.findViewById(R.id.rect);

        rect.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Animator animator =
                        ViewAnimationUtils.createCircularReveal(
                        rect,
                        0,
                        0,
                        0,
                        (float) Math.hypot(rect.getWidth(),
                                rect.getHeight()));
                animator.setInterpolator(
                        new AccelerateInterpolator());
                animator.setDuration(2000);
                animator.start();
            }
        });
    }
}
```

以上程序设置了两种形式，通过设置不同的坐标值，改变圆形展开的方式和效果，如图 12.22、图 12.23 所示。

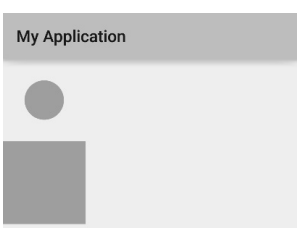

图 12.22 CircularReveal（中心）

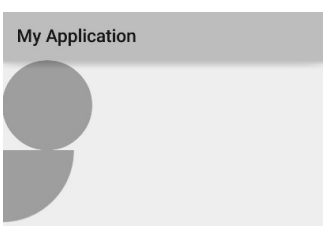

图 12.23 CircularReveal（边界）

12.8.3 View state changes Animation

在 Android 5.X 中，系统提供了视图状态改变来设置一个视图的状态切换动画。

- StateListAnimator

StateListAnimator 作为视图改变时的动画效果，通常会使用 Selector 来进行设置，但以前设置 Selector 的时候，通常是修改背景来达到反馈的效果。现在，在 Android 5.X 中，可以使用动画来作为视图改变的效果。

在 XML 中定义一个 StateListAnimator，并添加到 Selector 中，代码如下所示。

```
<?xml version="1.0" encoding="utf-8"?>
<selector xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:state_pressed="true">
        <set>
            <objectAnimator android:propertyName="rotationX"
                android:duration="@android:integer/config_shortAnimTime"
                android:valueTo="360"
                android:valueType="floatType"/>
        </set>
    </item>
    <item android:state_pressed="false">
        <set>
            <objectAnimator android:propertyName="rotationX"
                android:duration="@android:integer/config_shortAnimTime"
                android:valueTo="0"
                android:valueType="floatType"/>
        </set>
    </item>
</selector>
```

在一般的 XML 布局中，使用如下代码来将 StateListAnimator 添加给一个视图。

```
<Button
    android:layout_width="200dp"
    android:layout_height="200dp"
    android:stateListAnimator="@drawable/anim_change"/>
```

同样，在代码中也可以调用 AnimationInflater.loadStateListAnimator()方法，并且通过 View.setStateListAnimator()方法分配动画到视图上，效果如图 12.24 所示。

- animated-selector

animated-selector 同样是一个状态改变的动画效果 Selector。在 Android 5.0 中，很多 Material

Design 的控件设计，都是通过这种方式来实现的，例如我们熟悉的 check_box 的动画效果，就是使用类似帧动画的切换效果，模拟进行点击时的切换效果，如图 12.25 所示。

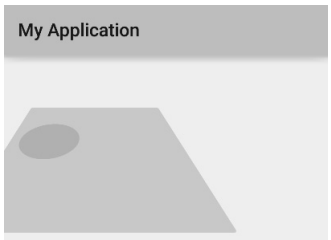

图 12.24　StateListAnimator

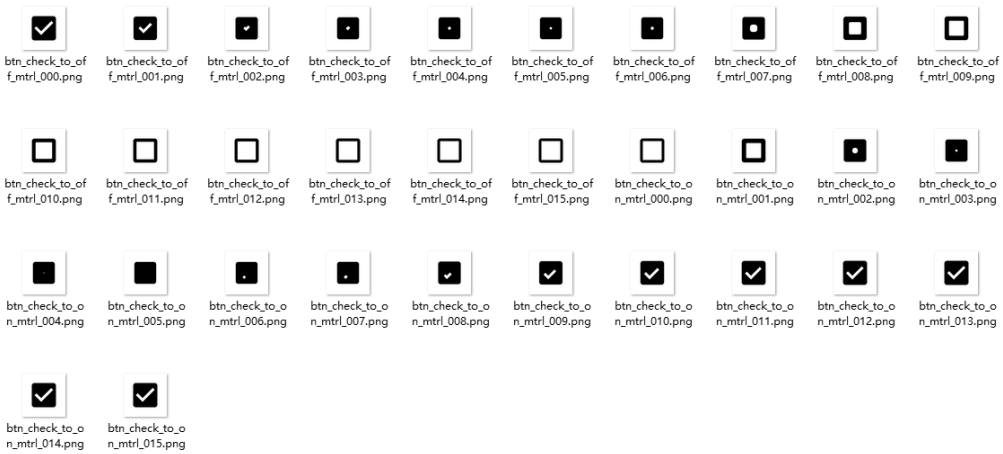

图 12.25　check_box 状态切换图

下面就仿照这个例子来实现一个具有动画效果的状态切换按钮。首先需要一组类似图 12.25 的状态切换图，如图 12.26 所示。

图 12.26　状态切换图

有了这样一组图，就可以在 XML 文件中定义 animated-selector。animated- selector 与 selector 的使用十分类似，同样是通过<item>标签来区分不同的状态，代码如下所示。

```
<item
    android:id="@+id/state_on"
    android:state_checked="true">
    <bitmap android:src="@drawable/ic_done_anim_000" />
</item>
<item android:id="@+id/state_off">
    <bitmap android:src="@drawable/ic_plus_anim_030" />
</item>
```

图 ic_done_anim_000 与图 ic_plus_anim_030 分别代表两种不同的状态。同时，我们也给这两种状态增加了 ID 来进行区分，下面就可以使用<transition>标签来给这两种状态设置不同的过渡图片，这点非常类似 Android 中的帧动画效果，完整代码如下所示。

```
<animated-selector xmlns:android="http://schemas.android.com/apk/
res/android">
    <item
        android:id="@+id/state_on"
        android:state_checked="true">
        <bitmap android:src="@drawable/ic_done_anim_030" />
    </item>
    <item android:id="@+id/state_off">
        <bitmap android:src="@drawable/ic_plus_anim_030" />
    </item>
    <transition
        android:fromId="@+id/state_on"
        android:toId="@+id/state_off">
        <animation-list>
            <item android:duration="16">
                <bitmap android:src="@drawable/ic_plus_anim_000" />
            </item>
            <item android:duration="16">
                <bitmap android:src="@drawable/ic_plus_anim_001" />
            </item>
            <item android:duration="16">
                <bitmap android:src="@drawable/ic_plus_anim_002" />
            </item>
            <item android:duration="16">
                <bitmap android:src="@drawable/ic_plus_anim_003" />
            </item>
            ……
```

```
                <item android:duration="16">
                    <bitmap android:src="@drawable/ic_plus_anim_028" />
                </item>
                <item android:duration="16">
                    <bitmap android:src="@drawable/ic_plus_anim_029" />
                </item>
                <item android:duration="16">
                    <bitmap android:src="@drawable/ic_plus_anim_030" />
                </item>
            </animation-list>

        </transition>
        <transition
            android:fromId="@+id/state_off"
            android:toId="@+id/state_on">
            <animation-list>
                <item android:duration="16">
                    <bitmap android:src="@drawable/ic_done_anim_000" />
                </item>
                <item android:duration="16">
                    <bitmap android:src="@drawable/ic_done_anim_001" />
                </item>
                <item android:duration="16">
                    <bitmap android:src="@drawable/ic_done_anim_002" />
                </item>
                <item android:duration="16">
                    <bitmap android:src="@drawable/ic_done_anim_003" />
                </item>
                ……
                <item android:duration="16">
                    <bitmap android:src="@drawable/ic_done_anim_028" />
                </item>
                <item android:duration="16">
                    <bitmap android:src="@drawable/ic_done_anim_029" />
                </item>
                <item android:duration="16">
                    <bitmap android:src="@drawable/ic_done_anim_030" />
                </item>
            </animation-list>
        </transition>
    </animated-selector>
```

有了 animated-selector 之后，只需要把它应用到一个 ImageView 上即可。同时，在代码中设置不同的点击状态。在 Android 中，通常使用如下所示的系统属性来设置切换状态。

```
    private static final int[] STATE_CHECKED = new int[]{
```

```
            android.R.attr.state_checked};
    private static final int[] STATE_UNCHECKED = new int[]{};
```

当点击时，通过 setImageState 方法来改变一个背景状态图，完整代码如下所示。

```
package com.imooc.animatedselector;

import android.app.Activity;
import android.graphics.drawable.Drawable;
import android.os.Bundle;
import android.view.View;
import android.widget.ImageView;

public class MainActivity extends Activity {

    private boolean mIsCheck;
    private static final int[] STATE_CHECKED = new int[]{
            android.R.attr.state_checked};
    private static final int[] STATE_UNCHECKED = new int[]{};
    private ImageView mImageView;
    private Drawable mDrawable;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        mImageView = (ImageView) findViewById(R.id.image);
        mDrawable = getResources().getDrawable(
                R.drawable.fab_anim);
        mImageView.setImageDrawable(mDrawable);
    }

    public void anim(View view) {
        if (mIsCheck) {
            mImageView.setImageState(STATE_UNCHECKED, true);
            mIsCheck = false;
        } else {
            mImageView.setImageState(STATE_CHECKED, true);
            mIsCheck = true;
        }
    }
}
```

程序运行效果如图 12.27 所示，初始状态时，按钮显示“+”。而当点击时，按钮显示“√”，同时具有一个切换效果，而不是直接从“+”变为“√”，如图 12.28 所示。

图 12.27　初始状态

图 12.28　点击状态

12.9　Toolbar

Toolbar 与 Actionbar 最大的区别就是 Toolbar 更加自由、可控。这也是 Google 在逐渐使用 Toolbar 替换 Actionbar 的原因，要使用 Toolbar 必须引入 appcompat-v7 支持，并设置主题为 NoActionBar，因此在 styles.xml 文件中，使用如下所示的代码进行设置。

```
<?xml version="1.0" encoding="utf-8"?>
<resources>

    <style name="AppTheme" parent="Theme.AppCompat.Light.NoActionBar">
        <!-- toolbar 颜色 -->
        <item name="colorPrimary">#4876FF</item>
        <!-- 状态栏颜色 -->

        <item name="colorPrimaryDark">#3A5FCD</item>
        <!-- 窗口的背景颜色 -->
        <item name="android:windowBackground">@android:color/white</item>
        <!-- add SearchView -->
        <item name="searchViewStyle">@style/MySearchView</item>
    </style>

    <style name="MySearchView" parent="Widget.AppCompat.SearchView" />
</resources>
```

在 Toolbar 中，可以像使用 Actionbar 那样增加一些小的 View，例如用于搜索的 SearchView。同时，不要忘了在 Gradle 配置文件中增加对 appcompat-v7 的引用，代码如下所示。

```
dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    compile 'com.android.support:appcompat-v7:21.0.3'
}
```

在代码中，可以通过如下所示的代码来添加 Toolbar 显示的标题和图标。

```
mToolbar = (Toolbar) findViewById(R.id.toolbar);
```

```
mToolbar.setLogo(R.drawable.ic_launcher);
mToolbar.setTitle("主标题");
mToolbar.setSubtitle("副标题");
setSupportActionBar(mToolbar);
```

菜单配置与 Actionbar 基本类似，代码如下所示。

```
<menu xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    tools:context=".MainActivity" >

    <item
        android:id="@+id/ab_search"
        android:orderInCategory="80"
        android:title="action_search"
        app:actionViewClass="android.support.v7.widget.SearchView"
        app:showAsAction="ifRoom"/>
    <item
        android:id="@+id/action_share"
        android:orderInCategory="90"
        android:title="action_share"
        app:actionProviderClass="android.support.v7.widget.ShareActionProvider"
        app:showAsAction="ifRoom"/>
    <item
        android:id="@+id/action_settings"
        android:orderInCategory="100"
        android:title="action_settings"
        app:showAsAction="never"/>

</menu>
```

一个典型的 Toolbar 在程序中的显示效果如图 12.29 所示。

图 12.29　Toolbar

通过 setSupportActionBar()方法，可以用 Toolbar 模拟出 Actionbar 的效果。下面我们将 Toolbar 与 DrawerLayout 结合起来，向大家演示 Android 5.X 中 DrawerToggle 状态变化的动态效果，即最左边的 Toggle 在 DrawerLayout 滑动时，会从三条横线渐变成一个箭头状的图案。

首先，在布局文件中引入 DrawerLayout，并为其设置显示的内容，代码如下所示。

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

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

    <android.support.v4.widget.DrawerLayout
        android:id="@+id/drawer"
        android:layout_width="match_parent"
        android:layout_height="match_parent" >

        <!-- 内容界面 -->
        <LinearLayout
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:background="@android:color/holo_blue_light"
            android:orientation="vertical" >
            <Button
                android:layout_width="100dp"
                android:layout_height="match_parent"
                android:text="内容界面"/>
        </LinearLayout>

        <!-- 侧滑菜单内容 必须指定其水平重力 -->
        <LinearLayout
            android:id="@+id/drawer_view"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:layout_gravity="start"
            android:orientation="vertical" >
            <Button
                android:layout_width="200dp"
                android:layout_height="match_parent"
                android:text="菜单界面"/>
        </LinearLayout>
    </android.support.v4.widget.DrawerLayout>
</LinearLayout>
```

接下来，在 Activity 中使用如下所示代码即可实现该效果。

```
getSupportActionBar().setDisplayHomeAsUpEnabled(true);
mDrawerLayout = (DrawerLayout) findViewById(R.id.drawer);
```

```
mDrawerToggle = new ActionBarDrawerToggle(
        this, mDrawerLayout, mToolbar,
        R.string.abc_action_bar_home_description,
        R.string.abc_action_bar_home_description_format);
mDrawerToggle.syncState();
mDrawerLayout.setDrawerListener(mDrawerToggle);
```

程序运行效果如图 12.30 所示，展开效果如图 12.31 所示。

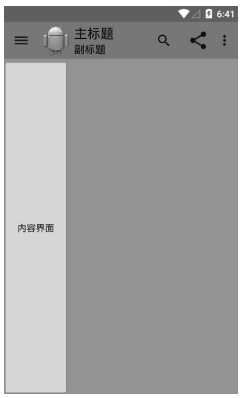

图 12.30 DrawerLayout 展开效果

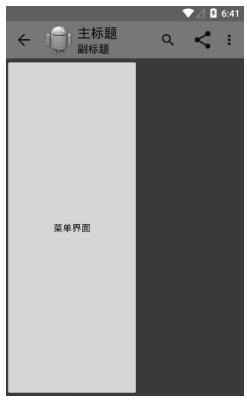

图 12.31　DrawerLayout 展开效果

12.10　Notification

Notification 作为一个事件触发通知型的交互提示接口，让我们可以在获得消息的时候，在状态栏、锁屏界面得到相应的提示信息。从 QQ、微信到各种推送通知、短信，这些 Notification 常常出现在状态栏。

Google 在 Android 5.0 上又进一步改进了通知栏，优化了 Notification。现在，在 Android 5.X 设备上，一个标准的 Notification 界面如图 12.32 所示。

当长按 Notification 的时候，会显示消息来源，如图 12.33 所示。

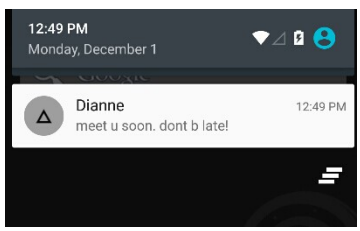

图 12.32　标准 Notification 界面

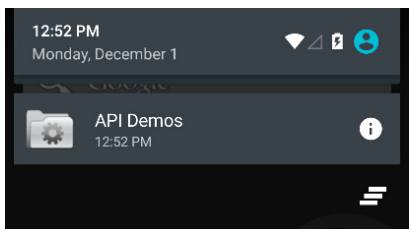

图 12.33　长按 Notification 后显示消息来源

Notification 会有一个从白色到灰色的动画切换效果，最终显示发出这个 Notification 的调用者。同时，在 Android 5.X 设备上，锁屏状态下我们也可以看见 Notification 通知了，如图 12.34 所示。

下面我们就分四重境界来看看该如何在 Android 5.0 下使用 Notification，整个示例程序的运行效果如图 12.35 所示。

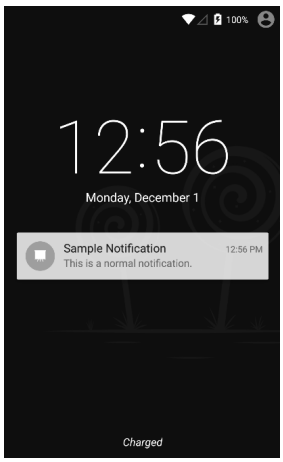

图 12.34　锁屏状态下的 Notification

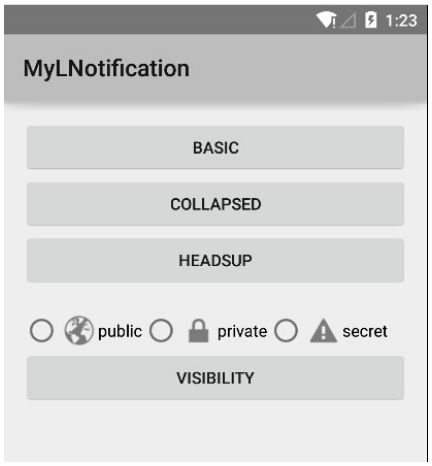

图 12.35　Notification 示例程序界面

12.10.1　基本的 Notification

通过 Notification.Builder 创建一个 Notification 的 builder，代码如下所示。

```
Notification.Builder builder = new Notification.Builder(this);
```

这个与 AlertDialog 的使用方法是不是非常相似呢？接下来，给点击 Notification 后要执行的操作增进一个 Intent，由于这个 Intent 不是马上就执行的，而是由用户触发的，所以 Android 给这样的 Intent 提供了一个 PendingIntent 来帮助我们完成这样的延迟操作。

PendingIntent 的使用非常简单，只需要在普通 Intent 的基础上包装一层就可以了，代码如下所示。

```
Intent intent = new Intent(Intent.ACTION_VIEW, Uri.parse("http://www.baidu.com"));
// 构造 PendingIntent
PendingIntent pendingIntent = PendingIntent.getActivity(this, 0, intent, 0);
```

这样当点击 Notification 之后，就会触发 PendingIntent 事件，打开浏览器开始浏览网页。

与使用 AlertDialog 一样，有了 builder 对象，可以给它增加各种属性。

```
builder.setSmallIcon(R.drawable.ic_launcher);
```

```
builder.setContentIntent(pendingIntent);
builder.setAutoCancel(true);
builder.setLargeIcon(BitmapFactory.decodeResource(getResources(),
R.drawable.ic_launcher));
builder.setContentTitle("Basic Notifications");
builder.setContentText("I am a basic notification");
builder.setSubText("it is really basic");
```

这些属性的具体含义，大家也不必一个个去仔细理解，结合后面演示的效果和命名，很容易就能理解了。最后一步，自然是将 Notification 显示出来，Android 系统通过 NotificationManager 系统服务来帮助我们管理 Notification，并通过调用 notify 方法来发出 Notification。

```
// 通过 NotificationManager 来发出 Notification
NotificationManager notificationManager =
        (NotificationManager) getSystemService(
        NOTIFICATION_SERVICE);
notificationManager.notify(NOTIFICATION_ID_BASIC,
        builder.build());
```

调用 NotificationManager 的 notify 方法时，需要传进去一个 ID。每个 Notification 都会有一个 ID，这个 ID 就是用来区分不同的 App 的 Notification 的。

通过以上很简单的几步，就完成了一个基本的 Notification。Notification 还可以配置 LED 灯和震动等选项，不过这些都只是增加一个属性罢了，这里就不详细演示了。最后来看看 Basic Notification 的效果如何，效果如图 12.36 所示。

相信结合运行出来的例子，大家应该可以很清楚地理解上面配置的参数的意义了。

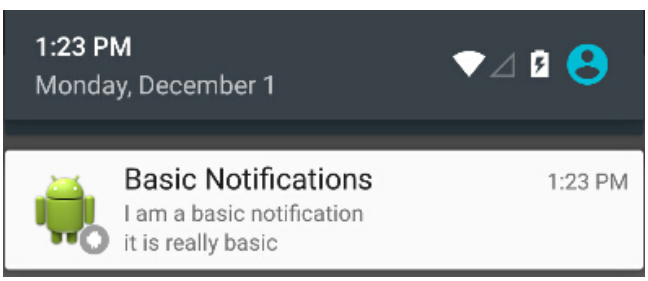

图 12.36　Basic Notification 显示效果

12.10.2　折叠式 Notification

折叠式 Notification 也是一种自定义视图的 Notification，常常用于显示长文本。它拥有两个是视图状态，一个是普通状态下的视图状态，另一个是展开状态下的视图状态。在 Notification 中，使用 RemoteViews 来帮助我们创建一个自定义的 Notification 视图，代码如下所示。

```
// 通过 RemoteViews 来创建自定义的 Notification 视图
RemoteViews contentView =
        new RemoteViews(getPackageName(), R.layout.notification);
contentView.setTextViewText(R.id.textView, "show me when collapsed");
```

Notification 的布局如下所示。

```
<?xml version="1.0" encoding="utf-8"?>

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:gravity="center_horizontal">

    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:id="@+id/textView"
        android:textColor="#ff43aebe"
        android:gravity="center" />

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

</LinearLayout>
```

通过如下代码，就可以将一个视图指定为 Notification 正常状态下的视图。

```
notification.contentView = contentView;
```

将另一个展开的布局通过如下代码指定为展开时的视图。

```
notification.bigContentView = expandedView;
```

完整的代码如下所示。

```
Intent intent = new Intent(Intent.ACTION_VIEW,
        Uri.parse("http://www.sina.com"));
PendingIntent pendingIntent = PendingIntent.getActivity(this,0,intent,0);
Notification.Builder builder = new Notification.Builder(this);
builder.setSmallIcon(R.drawable.ic_launcher);
builder.setContentIntent(pendingIntent);
```

```
builder.setAutoCancel(true);
builder.setLargeIcon(BitmapFactory.decodeResource(
        getResources(), R.drawable.ic_launcher));
// 通过 RemoteViews 来创建自定义的 Notification 视图
RemoteViews contentView =
        new RemoteViews(getPackageName(),
                R.layout.notification);
contentView.setTextViewText(R.id.textView,
        "show me when collapsed");

Notification notification = builder.build();
// 指定视图
notification.contentView = contentView;
// 通过 RemoteViews 来创建自定义的 Notification 视图
RemoteViews expandedView =
        new RemoteViews(getPackageName(),
                R.layout.notification_expanded);
// 指定视图
notification.bigContentView = expandedView;

NotificationManager nm = (NotificationManager)
        getSystemService(NOTIFICATION_SERVICE);
nm.notify(NOTIFICATION_ID_COLLAPSE, notification);
```

折叠时，显示效果如图 12.37 所示。

展开时，显示效果如图 12.38 所示。

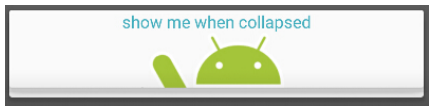

图 12.37　折叠时 Notification 视图

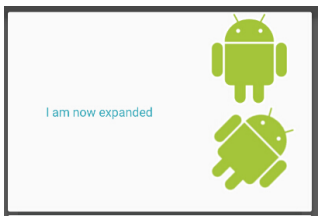

图 12.38　展开时 Notification 视图

12.10.3　悬挂式 Notification

悬挂式 Notification 是在 Android 5.0 中新增加的方式，Google 希望通过这种方式来给用户带来更好的体验。这种被称为 Headsup 的 Notification 方式，可以在屏幕上方产生 Notification 且不会打断用户操作，能给用户以 Notification 形式的通知。

在 Android Sample 中，Google 给我们展示了如何完成这样一个效果，代码如下所示。

```
Notification.Builder builder = new Notification.Builder(this)
```

```
            .setSmallIcon(R.drawable.ic_launcher)
            .setPriority(Notification.PRIORITY_DEFAULT)
            .setCategory(Notification.CATEGORY_MESSAGE)
            .setContentTitle("Headsup Notification")
            .setContentText("I am a Headsup notification.");

    Intent push = new Intent();
    push.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
    push.setClass(this, MainActivity.class);
    PendingIntent pendingIntent = PendingIntent.getActivity(
            this, 0, push, PendingIntent.FLAG_CANCEL_CURRENT);
    builder.setContentText("Heads-Up Notification on Android 5.0")
            .setFullScreenIntent(pendingIntent, true);

    NotificationManager nm = (NotificationManager)
            getSystemService(NOTIFICATION_SERVICE);
    nm.notify(NOTIFICATION_ID_HEADSUP, builder.build());
```

如以上代码所示，通过 setFullScreenIntent，我们很轻松地将一个 Notification 变成了悬挂式 Notification，程序显示效果如图 12.39 所示。

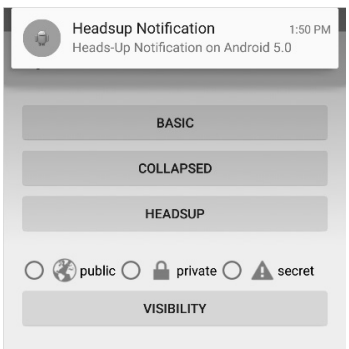

图 12.39　悬挂式 Notification

12.10.4　显示等级的 Notification

最后一重境界也是在 Android 5.X 中新加入的一种模式——Notification 的显示等级。Android 5.X 将 Notification 分成了三个等级。

- VISIBILITY_PRIVATE——表明只有当没有锁屏的时候会显示
- VISIBILITY_PUBLIC——表明在任何情况下都会显示
- VISIBILITY_SECRET——表明在 pin、password 等安全锁和没有锁屏的情况下才能够显示

设置 Notification 等级的方式非常简单，同样是借助 builder 对象，代码如下所示。

```
builder.setVisibility(Notification.VISIBILITY_PUBLIC);
```

在前面的示例代码中，我们使用了一个 RadioGroup 来演示 VISIBILITY 等级，实现代码如下所示。

```
RadioGroup radioGroup = (RadioGroup) findViewById(
        visibility_radio_group);
Notification.Builder builder = new Notification.Builder(this)
        .setContentTitle("Notification for Visibility Test");
switch (radioGroup.getCheckedRadioButtonId()) {
    case R.id.radio_button_public:
        builder.setVisibility(Notification.VISIBILITY_PUBLIC);
        builder.setContentText("Public");
        builder.setSmallIcon(R.drawable.ic_public);
        break;
    case R.id.radio_button_private:
        builder.setVisibility(Notification.VISIBILITY_PRIVATE);
        builder.setContentText("Private");
        builder.setSmallIcon(R.drawable.ic_private);
        break;
    case R.id.radio_button_secret:
        builder.setVisibility(Notification.VISIBILITY_SECRET);
        builder.setContentText("Secret");
        builder.setSmallIcon(R.drawable.ic_secret);
        break;
}
NotificationManager nm = (NotificationManager)
        getSystemService(NOTIFICATION_SERVICE);
nm.notify(NOTIFICATION_ID_VISIBILITY, builder.build());
```

代码结构非常清晰，通过 build 对象的 setVisibility 方法，就可以轻松地设置 Notification 显示等级了。

Notification 在 Android 5.X 中的改动非常多，这里只是让大家有一个初步的概念，还有其他的改动，比如以下两种。

增加了设置 Notification 背景颜色的接口，代码如下所示。

```
builder.setColor(Color.RED)
```

增加了设置 Notification 的 category 接口，category 用来确定 Notification 显示的位置，参数就是各种 category 的类型，代码如下所示。

```
builder .setCategory(Notification.CATEGORY_MESSAGE)
```