第 13 章

13 使用MediaPlayer播放音频

接下来的三章，我们先暂停CriminalIntent应用的开发，转而开发另一个应用。使用MediaPlayer类，新应用可支持播放一段历史事件的音频文件，如图13-1所示。

图13-1　你好，月球！

MediaPlayer是一个支持音频及视频文件播放的Android类，可播放不同来源（本地或网络流媒体）、多种格式（如WAV、MP3、Ogg Vorbis、MPEG-4以及3GPP）的多媒体文件。

创建一个名为HelloMoon的项目。在新应用向导对话框中，选择Holo Dark作为应用的主题，如图13-2所示。

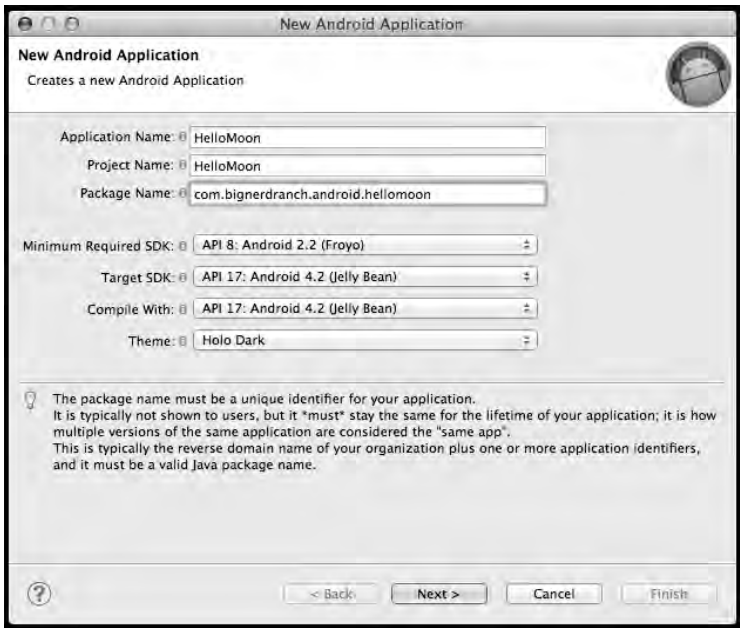

图13-2　使用Holo Dark主题创建HelloMoon应用

单击Next按钮继续。该应用无定制启动器图标，记得清除相关选项。最后选择使用空白activity模板，并通过模板创建一个名为HelloMoonActivity的activity。

13.1　添加资源

HelloMoon应用需要一个图片文件以及一个音频文件。这两个文件包含在本书随书代码文件中（http://www.bignerdranch.com/solutions/AndroidProgramming.zip）。下载随书代码文件包后，在以下路径找到它们：

- 13_Audio/HelloMoon/res/drawable-mdpi/armstrong_on_moon.jpg
- 13_Audio/HelloMoon/res/raw/one_small_step.wav

HelloMoon是个简单的小应用，按照Android屏幕密度基准（~160dpi），我们仅创建了一个符合该基准的图片文件（armstrong_on_moon.jpg）。将armstrong_on_moon.jpg复制至drawable-mdpi目录下。

音频文件将会放置在res/raw目录下。目录raw负责存放那些不需要Android编译系统特别处理的各类文件。

项目中的res/raw目录并非默认存在，因此必须手工添加它。（右键单击res目录，选择New → Folder菜单项。）然后将one_small_step.wav文件复制到新建目录下。

（也可顺便将13_Audio/HelloMoon/res/raw/apollo_17_stroll.mpg复制到res/raw目录下。本章末尾的挑战练习将会用到它。）

最后，打开res/values/strings.xml，对照代码清单13-1，添加HelloMoon应用所需的字符串资源。

代码清单13-1 添加字符串资源（strings.xml）

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

  <string name="app_name">HelloMoon</string>
  <string name="hello_world">Hello world!</string>
  <string name="menu_settings">Settings</string>
  <string name="hellomoon_play">Play</string>
  <string name="hellomoon_stop">Stop</string>
  <string name="hellomoon_description">Neil Armstrong stepping
          onto the moon</string>

</resources>
```

（HelloMoon应用为什么在按钮上使用“Play”和“Stop”字符串资源而非图标文件？第15章我们将学习应用本地化的相关内容。使用图标文件会增加本地化的难度。）

准备完必需的资源文件，下面我们来看看HelloMoon应用的整体设计图，如图13-3所示。

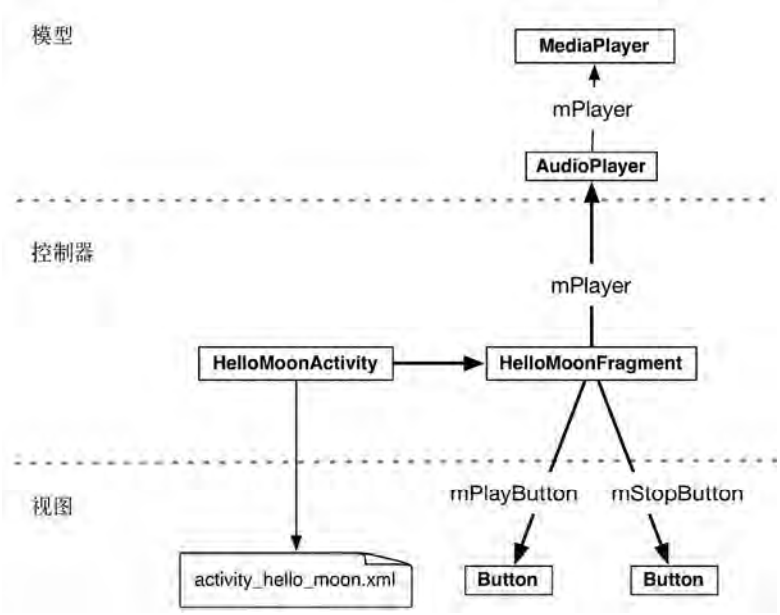

图13-3 HelloMoon应用的对象图解

图中可以看出，HelloMoon应用包含一个HelloMoonActivity及其托管的HelloMoonFragment。

AudioPlayer是我们编写的类，用于封装MediaPlayer类。也可选择不封装MediaPlayer类，而让HelloMoonFragment直接与MediaPlayer进行交互。不过，为保持代码的整洁与独立，我们推荐封装MediaPlayer类的设计。

创建AudioPlayer类之前，先来完成应用开发的其他部分。经过前几章的学习，我们应该已

经掌握了以下基本的应用开发步骤：

- 定义fragment的布局
- 创建fragment类
- 修改activity及其布局，实现对fragment的托管

13.2　定义 HelloMoonFragment 布局文件

以TableLayout为根元素，新建一个名为fragment_hello_moon.xml的布局文件。

参照图13-4，完成fragment_hello_moon.xml布局文件的定义。

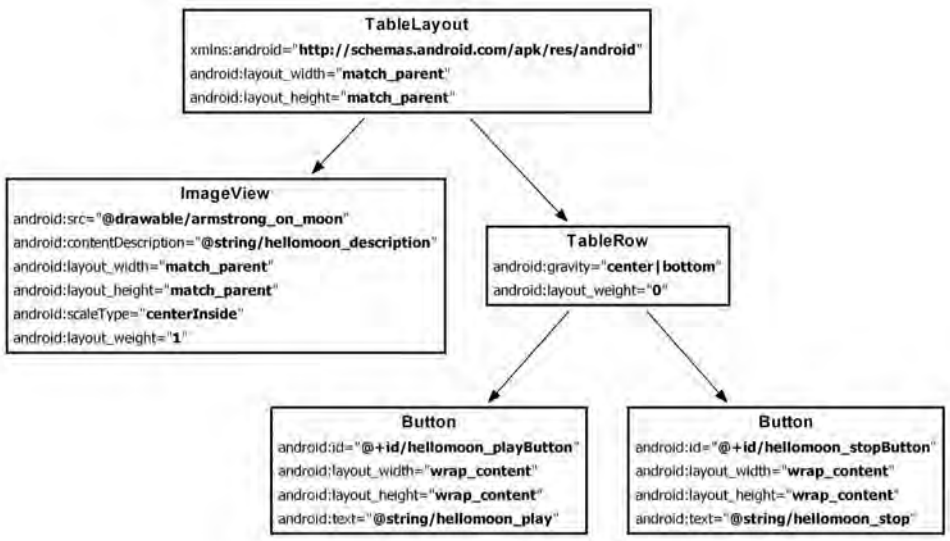

图13-4　HelloMoon应用的布局图解

TableLayout使用起来和LinearLayout差不多。可使用TableRow，而非嵌套使用的LinearLayout，来布置组件。联合使用TableLayout与TableRow，可更容易地布置形成排列整齐的视图。在HelloMoon应用里，TableLayout能够协助将两个按钮并排放置到同样大小的两列中。

为什么没有将ImageView也放入TableRow中呢？TableRow子组件的行为方式类似于表里的单元格。这里我们希望让ImageView占据整个屏幕。如果将ImageView也定义为TableRow的子组件，则TableLayout也会让列中其他单元格占据整个屏幕。而作为TableLayout的直接子组件，ImageView可自由地按需配置显示，完全不影响两个按钮以等宽的两列并排显示。

注意，TableRow组件无需声明高度和宽度的属性定义。实际上，它使用的是TableLayout的高度和宽度属性定义及其所有其他属性定义。不过，从另一个角度来看，嵌套的LinearLayout可以更灵活地布置并显示组件。

在图形化工具中预览布局，看看应用背景是什么颜色。新建项目时，我们选择了Holo Dark作为应用的主题。然而，在本书写作之时，新建向导仍会忽略主题选项，而选择使用浅白主题。接下来，我们来看看如何修正该问题。

（如果主题背景是黑色的，这说明向导功能问题已被修正，因此也无需按照下面小节的描述手动重置应用主题了。）

手动重置应用主题

以下代码可以看出，应用的主题是在配置文件的application元素节点下声明的：

```
...
  <application
    android:allowBackup="true"
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme" >
    ...
  </application>

</manifest>
```

android:theme是非强制性使用属性。如果配置文件中没有声明主题，应用则会使用设备的默认主题。

以上代码可以看出，已声明主题的属性值@style/AppTheme实际是对其他资源的引用。在包浏览器中，找到并打开res/values/styles.xml文件。找到名为AppBaseTheme的style元素，将其parent属性值修改为android:Theme，如代码清单13-2所示。

代码清单13-2 修改默认的style文件（res/values/styles.xml）

```
<style name="AppBaseTheme" parent="android:Theme.Light">
<style name="AppBaseTheme" parent="android:Theme">
```

在res资源目录中，还有两个有修饰后缀的values目录，各目录下还含有一个styles.xml文件。values目录的修饰后缀指的是API级别。保存在res/values-v11/styles.xml文件中的属性值适用于API 11-13级，而保存在res/values-v14/styles.xml文件中的属性值则适用于API 14级或更高的级别。

打开res/values-v11/styles.xml文件，找到名为AppBaseTheme的style元素，并将其parent属性值修改为android:Holo。我们希望所有运行API 11级及更高级别的设备都使用这个主题，因此res/values-v14/目录也就不再需要了。在包浏览器中，将该目录从HelloMoon项目中删除。

保存所有修改过的文件，然后再次预览布局页面。可以看到，布局有了与图片文件完全匹配的黑色背景。

13.3 创建 HelloMoonFragment

创建一个名为`HelloMoonFragment`的类，设置其超类为`android.support.v4.app.Fragment`。

覆盖`HelloMoonFragment.onCreateView(...)`方法，实例化布局文件，并引用按钮，如代码清单13-3所示。

代码清单13-3　初步配置HelloMoonFragment类（HelloMoonFragment.java）

```
public class HelloMoonFragment extends Fragment {

    private Button mPlayButton;
    private Button mStopButton;

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup parent,
            Bundle savedInstanceState) {
        View v = inflater.inflate(R.layout.fragment_hello_moon, parent, false);

        mPlayButton = (Button)v.findViewById(R.id.hellomoon_playButton);
        mStopButton = (Button)v.findViewById(R.id.hellomoon_stopButton);

        return v;
    }
}
```

13.4　使用布局 fragment

在CriminalInten应用中，我们一直是通过在activity代码中添加fragment的方式来实现其托管的。而在HelloMoon应用中，我们将使用布局fragment。使用布局fragment，即在fragment元素节点中指定fragment的类。

打开activity_hello_moon.xml文件，以代码清单13-4所示的fragment元素替换原有内容。

代码清单13-4　创建布局fragment（activity_hello_moon.xml）

```
<?xml version="1.0" encoding="utf-8"?>
<fragment xmlns:android="http://schemas.android.com/apk/res/android"
  android:id="@+id/helloMoonFragment"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:name="com.bignerdranch.android.hellomoon.HelloMoonFragment">

</fragment>
```

运行HelloMoon应用前，修改HelloMoonActivity类的超类为FragmentActivity，如代码清单13-5所示。这是HelloMoonActivity类代码所需做出的唯一修改。

代码清单13-5　让HelloMoonActivity继承FragmentActivity类（HelloMoonActivity.java）

```
public class HelloMoonActivity extends ~~Activity~~ FragmentActivity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_hello_moon);
    }
}
```

运行HelloMoon应用。可看到HelloMoonActivity托管了HelloMoonFragment的视图。

以上便是托管一个布局fragment的全部操作。总结来说就是，在布局中指定fragment类，随后通过布局它被添加给activity并显示在屏幕上。以下为布局fragment的后台工作机制。

HelloMoonActivity在调用.setContentView(...)方法，并实例化activity_hello_moon.xml布局时，发现了fragment元素。于是，FragmentManager接着就创建了HelloMoonFragment的一个实例，并将其添加到fragment队列中。然后，它调用HelloMoonFragment的onCreateView(...)方法，并将该方法返回的视图放置到activity布局中fragment元素占据的位置上，如图13-5所示。

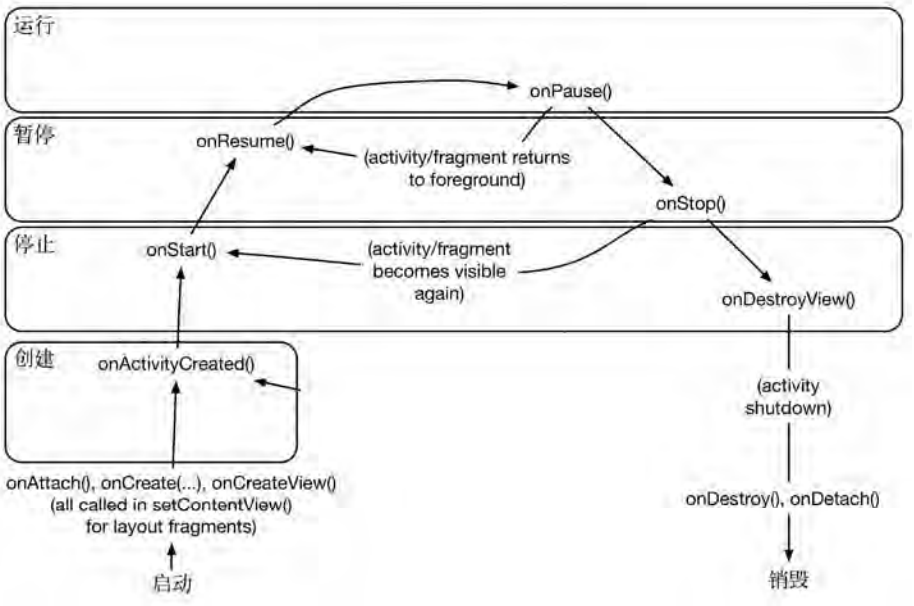

图13-5 布局fragment的生命周期

有得必有失，使用如此简单的方法托管fragment，同时也失去了只有显式地使用FragmentManager才能获得的灵活性和掌控能力。

- 可覆盖fragment的生命周期方法，以响应各种事件。但无法控制调用这些方法的时机。
- 无法提交移除、替换、分离布局fragment的事务。activity被创建后，即无法做出任何改变。
- 无法附加argument给布局fragment。附加argument必须在fragment创建后并被添加给FragmentManager之前完成。如果使用布局fragment，这些事件何时发生，我们无从得知。

虽然存在以上问题，但对于简单应用或复杂应用的某些静态部分而言，使用布局fragment是一个不错的选择。

HelloMoonFragment的托管完成了。下面，我们来处理应用的音频播放部分。

13.5 音频播放

在com.bignerdranch.android.hellomoon包中，创建一个名为AudioPlayer的新类，保持超类java.lang.Object不变。

在AudioPlayer.java中，添加存放MediaPlayer实例的成员变量，以及停止和播放该实例的方

法，如代码清单13-6所示。

代码清单13-6　使用MediaPlayer类的简单播放代码（AudioPlayer.java）

```
public class AudioPlayer {

    private MediaPlayer mPlayer;

    public void stop() {
        if (mPlayer != null) {
            mPlayer.release();
            mPlayer = null;
        }
    }

    public void play(Context c) {
        mPlayer = MediaPlayer.create(c, R.raw.one_small_step);
        mPlayer.start();
    }
}
```

在play(Context)方法中，选择调用MediaPlayer.create(Context, int)方法。MediaPlayer需利用传入的Context来寻找音频文件的资源ID。（如果音频来自于其他渠道，如网络或本地URI，则应使用其他MediaPlayer.create(...)方法。）

在AudioPlayer.stop()方法中，释放MediaPlayer实例并将mPlayer变量设置为null。调用MediaPlayer.release()方法，可销毁该实例。

销毁是“停止”的一种具有攻击意味的说法，但我们有充足的理由使用销毁一词。除非调用MediaPlayer.release()方法，否则MediaPlayer将一直占用着音频解码硬件及其他系统资源。而这些资源是由所有应用共享的。MediaPlayer类有一个stop()方法。该方法可使MediaPlayer实例进入停止状态，等需要时再重新启动。不过，对于简单的音频播放应用，建议使用release()方法销毁实例，并在需要时进行重建。

基于以上原因，有一个简单可循的规则：只保留一个MediaPlayer实例，保留的时长即音频文件播放的时长。

为强化该规则，我们来修改play(Context)方法。初始调用一个stop()方法，再设置一个监听器监听音频播放，音频文件播放一完成，就调用stop()方法，如代码清单13-7所示。

代码清单13-7　避免多MediaPlayer实例（AudioPlayer.java）

```
...

public void play(Context c) {
    stop();

    mPlayer = MediaPlayer.create(c, R.raw.one_small_step);

    mPlayer.setOnCompletionListener(new MediaPlayer.OnCompletionListener() {
        public void onCompletion(MediaPlayer mp) {
            stop();
        }
    });
```

```
            mPlayer.start();
        }

    }
```

在play(Context)方法的开头就调用stop()方法，可避免用户多次单击Play按钮创建多个MediaPlayer实例的情况发生。音频文件完成播放后，立即调用stop()方法，可尽可能快地释放MediaPlayer实例及其占用的资源。

fragment被销毁后，还需在HelloMoonFragment中调用AudioPlayer.stop()方法，以避免MediaPlayer的不停播放。在HelloMoonFragment中，覆盖onDestroy()方法，从而实现对AudioPlayer.stop()方法的调用，如代码清单13-8所示。

代码清单13-8 覆盖onDestroy()方法（HelloMoonFragment.java）

```
        ...

        @Override
        public void onDestroy() {
            super.onDestroy();
            mPlayer.stop();
        }
    }
```

HelloMoonFragment被销毁后，MediaPlayer仍可不停地播放，这是因为MediaPlayer运行在一个不同的线程上。我们会在第26章中学习到更多有关线程管理的知识，所以这里暂不对HelloMoon应用的多线程使用进行介绍。

关联并设置play和stop按钮

返回HelloMoonFragment.java，创建AudioPlayer实例并设置play和stop按钮的监听器方法，即可实现音频的播放，如代码清单13-9所示。

代码清单13-9 关联并设置Play按钮（HelloMoonFragment.java）

```
public class HelloMoonFragment extends Fragment {
    private AudioPlayer mPlayer = new AudioPlayer();
    private Button mPlayButton;
    private Button mStopButton;

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup parent,
            Bundle savedInstanceState) {
        View v = inflater.inflate(R.layout.fragment_hello_moon, parent, false);

        mPlayButton = (Button)v.findViewById(R.id.hellomoon_playButton);
        mPlayButton.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v) {
                mPlayer.play(getActivity());
            }
        });

        mStopButton = (Button)v.findViewById(R.id.hellomoon_stopButton);
        mStopButton.setOnClickListener(new View.OnClickListener() {
```

```
            public void onClick(View v) {
                mPlayer.stop();
            }
        });
        return v;
    }

}
```

运行HelloMoon应用。单击Play按钮，欣赏一段历史事件记录音频。

本章介绍的内容仅是MediaPlayer功能的冰山一角。可查阅Android开发者网站上的MediaPlayer开发者手册（http://developer.android.com/guide/topics/media/mediaplayer.html），了解更多相关内容。

13.6　挑战练习：暂停音频播放

为用户提供暂停音频播放的功能。请查阅MediaPlayer类的参考手册寻找相关操作方法。

13.7　深入学习：播放视频

关于视频的播放，Android提供了多种实现方式。其一便是使用刚才讲到的MediaPlayer，而我们唯一要做的就是设置在哪里播放视频。

在Android系统中，快速刷新显示的可视图像（如视频）是在SurfaceView中显示的。准确地说，是在SurfaceView内嵌的Surface中显示的。通过获取SurfaceView的SurfaceHolder，可实现在Surface上显示视频。我们会在第19章中就相关内容做详细介绍。而现在只需知道调用MediaPlayer.setDisplay(SurfaceHolder)方法，将MediaPlayer类与SurfaceHolder关联起来即可。

通常来说，使用VideoView实例播放视频更容易些。不同于SurfaceView同MediaPlayer的交互，VideoView是与MediaController交互的，这样可以方便地提供视频播放界面。

使用VideoView唯一棘手的地方是，它并不接受资源ID，而只接受文件路径或Uri对象。要创建一个指向Android资源的Uri，可使用以下代码：

```
Uri resourceUri = Uri.parse("android.resource://" +
  "com.bignerdranch.android.hellomoon/raw/apollo_17_stroll");
```

以上代码可看出，我们需使用android.resource格式、用作主机名的包名、资源类型以及资源名称组成一个路径以创建一个URI。完成后，将它传递给VideoView使用。

13.8　挑战练习：在 HelloMoon 应用中播放视频

增强HelloMoon应用的功能，以支持播放视频文件。如还未获取apollo_17_stroll.mpg视频文件，请从随书代码文件的对应章节目录中找到该视频，并复制至项目的res/raw目录中。然后，使用深入学习部分介绍的方法编写代码，实现视频的播放。