第3章 Activity和界面布局

Activity是Android应用程序最主要的展示窗口。本章首先介绍Android应用的组成和有关Activity的基础知识;其次介绍基于XML文件完成Activity布局的方法、在Activity中通过Java编程方式设定布局的方法,以及Android的资源管理与使用方法;最后介绍常用的布局方式,内容涉及线性布局、相对布局、表格布局、网格布局、帧布局、约束布局。

3.1 Activity及其生命周期

3.1.1 Android应用的基本组件

一般来说,Android应用程序由Activity、ContentProvider、Service、BroadcastReceiver等组成。当然,有些应用程序可能只包含其中部分而非全部。它们在AndroidManifest.xml清单文件中以不同的XML标签声明后,才可以在应用程序中使用。

1. Activity

Activity一般含有一组用于构建用户界面(UI)的Widget控件,如按钮Button、文本框EditText、列表ListView等,实现与用户的交互,相当于Windows应用程序的窗口或网络应用程序的Web页面。一个功能完善的Android应用程序一般由多个Activity构成,这些Activity之间可互相跳转,可进行页面间的数据传递。例如,显示一个E-mail通讯簿列表的界面就是一个Activity,而编辑通讯簿界面则是另一个Activity。

2. ContentProvider

ContentProvider是Android系统提供的一种标准的数据共享机制。在Android平台下,一个应用程序使用的数据存储都是私有的,其他应用程序是不能访问和使用的。私有数据可以是存储在文件系统中的文件,也可以是SQLite中的数据库。当需要共享数据时,ContentProvider提供了应用程序之间数据交换的机制。一个应用程序通过实现一个ContentProvider的抽象接口将自己的数据暴露出去,并且隐蔽了具体的数据存储实现,这样既实现了应用程序内部数据的保密性,又能够让其他应用程序使用这些私有数据,同时实现了权限控制,保护了数据交互的安全性。

3. Service

Service 是相对于 Activity 独立且可以保持后台运行的服务，相当于一个在后台运行的没有界面的 Activity。如果应用程序并不需要显示交互界面但却需要长时间运行，就需要使用 Service。例如，在后台运行的音乐播放器，为了避免音乐播放器在后台运行时被终止而停播，需要为其添加 Service，通过调用 Context.startService()方法，让音乐播放器一直在后台运行，直到使用者再调出音乐播放器界面并关掉它为止。用户可以通过 StartService()方法启动一个 Service，也可以通过 bindService()方法来绑定一个 Service 并启动它。

4. BroadcastReceiver

在 Android 系统中，广播是一种广泛运用的在应用程序之间传输信息的机制。而 BroadcastReceiver 是用来接收并响应广播消息的组件，不包含任何用户界面。它可以通过启动 Activity 或者 Notification 通知用户接收重要信息。Notification 能够通过多种方法提示用户，包括闪动背景灯、振动设备、发出声音或在状态栏上放置一个持久的图标。

Activity、Service 和 BroadcastReceiver 都是由 Intent 异步消息激活的。Intent 用于连接以上各个组件，并在其间传递消息。例如，广播机制一般通过下述过程实现：首先在需要发送信息的地方，把要发送的信息和用于过滤的信息(如 Action、Category)装入一个 Intent 对象，然后通过调用 Context.sendBroadcast()或 sendOrderBroadcast()方法，把 Intent 对象以广播方式发送出去。Android 系统会对所有已经注册的 BroadcastReceiver 检查其 Intent 过滤器(IntentFilter)是否与发送的 Intent 相匹配，若匹配就会调用其 onReceive()方法，接收广播并响应。例如，对于一个电话程序，当有来电时，电话程序就自动使用 BroacastReceiver 取得对方的来电消息并显示。使用 Intent 还可以方便地实现各个 Activity 间的跳转和数据传递。

3.1.2　什么是 Activity

Activity 是 Android 四大组件中最基本的组件，是 Android 应用程序中最常用也是最重要的部分。在应用程序中，用户界面主要通过 Activity 呈现，包括显示控件、监听和处理用户的界面事件并做出响应。Activity 在界面上的表现形式有全屏窗体、非全屏悬浮窗体、对话框等。在模拟器上运行应用程序时，可以按 HOME 键或回退键退出当前 Activity。

对于大多数与用户交互的程序来说，Activity 是必不可少的，也是非常重要的。刚开始接触 Android 应用程序时，可以暂且将 Activity 简单地理解为用户界面。新建一个 Android 项目时，系统默认生成一个启动的主 Activity，其默认的类名为 MainActivity，源码文件中的主要内容如代码段 3-1 所示。

代码段 3-1　MainActivity 源代码

```
package edu.hebust.zxm.myfirstapplication;
import androidx.appcompat.app.AppCompatActivity;
                                    //AppCompatActivit 是 android.app.Activity 的子类
```

```
import android.os.Bundle;          //用于映射字符串值
public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
              //设置布局,它加载了 res/layout/activity_main.xml 中定义的界面元素
    }
}
```

应用程序中的每个 Activity 都必须继承自 android.app.Activity 类并重写(Override)其 OnCreate()方法。

Activity 通常要与布局资源文件(res/layout 目录下的 XML 文件)相关联,并通过 setContentView()方法将布局呈现出来。在 Activity 类中通常包含布局控件的显示、界面交互设计、事件的响应设计以及数据处理设计、导航设计等内容。

一个 Android 应用程序可以包含一个或多个 Activity,一般在程序启动后会首先呈现一个主 Activity,用于提示用户程序已经正常启动并显示一个初始的用户界面。需要注意的是,应用程序中的所有 Activity 都必须在 AndroidManifest.xml 文件中添加相应的声明,并根据需要设置其属性和<intent-filter>。例如,代码段 3-2 含有对两个 Activity(MainActivity 和 SecondActivity)的声明,代码中有两个<activity>元素,第一个为系统默认生成的 MainActivity,第二个是新建的 SecondActivity,其中的 MainActivity 是程序入口。

代码段 3-2　AndroidManifest.xml 文件中的声明

```
<application
    android:allowBackup="true"
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme" >
    <activity
        android:name="edu.hebust.zxm.myfirstapplication.MainActivity"
        android:exported="true" >
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>
    <activity
        android:name="edu.hebust.zxm.myfirstapplication.SecondActivity"
        android:label="SecondActivity" >
    </activity>
</application>
```

3.1.3　Activity 的生命周期

所有 Android 组件都具有自己的生命周期，生命周期是指从组件建立到组件销毁的整个过程。在生命周期中，组件会在可见、不可见、活动、非活动等状态中不断变化。

Activity 的生命周期指 Activity 从启动到销毁的过程。生命周期由系统控制，程序无法改变，但可以用 onSaveInstanceState()方法保存其状态。了解 Activity 的生命周期有助于理解 Activity 的运行方式和编写正确的 Activity 代码。

Activity 在生命周期中表现为 4 种状态，分别是活动状态、暂停状态、停止状态和非活动状态。处于活动状态时，Activity 在用户界面中处于最上层，能完全被用户看到，并与用户进行交互。处于暂停状态时，Activity 在界面上被部分遮挡，该 Activity 不再位于用户界面的最上层，且不能与用户进行交互。处于停止状态时，Activity 在界面上完全不能被用户看到，也就是说这个 Activity 被其他 Activity 全部遮挡。非活动状态指不在以上 3 种状态中的 Activity。

参考 Android SDK 官网文档中的说明，Activity 生命周期如图 3-1 所示。该示意图中涉及的方法被称为生命周期方法，当 Activity 状态发生改变时，相应的方法会被自动调用。

android.app.Activity 类是 Android 提供的基类，应用程序中的每个 Activity 都继承自该类，通过重写父类的生命周期方法来实现自己的功能。在代码段 3-1 中，@Override 表示重写父类的 onCreate()方法，Bundle 类型的参数保存了应用程序上次关闭时的状态，并且可以通过一个 Activity 传递给下一个 Activity。在 Activity 的生命周期中，只要离开了可见阶段(即失去了焦点)，它就很可能被进程终止，这时就需要一种机制能保存当时的状态，这就是其参数 savedInstanceState 的作用。

(1) 启动 Activity 时，系统会先调用其 onCreate()方法，然后调用 onStart()方法，最后调用 onResume()方法，Activity 进入活动状态。

(2) 当 Activity 被其他 Activity 部分遮盖或被锁屏时，Activity 不能与用户交互，会调用 onPause()方法，进入暂停状态。

(3) 当 Activity 由被遮盖回到前台或解除锁屏时，会调用 onResume()方法，再次进入活动状态。

(4) 当切换到新的 Activity 界面或按 Home 键回到主屏幕时，当前 Activity 完全不可见，转到后台。此时会先调用 onPause()方法，然后调用 onStop()方法，Activity 进入停止状态。

(5) 当 Activity 处于停止状态时，用户后退回到此 Activity，会先调用 onRestart()方法，然后调用 onStart()方法，最后调用 onResume()方法，再次进入运行状态。

(6) 当 Activity 处于被遮盖状态或者后台不可见，即处于暂停状态或停止状态时，如果系统内存不足，就有可能杀死这个 Activity。而后用户如果返回 Activity，则会再依次调用 onCreate()方法、onStart()方法、onResume()方法，使其进入活动状态。

(7) 用户退出当前 Activity 时，先调用 onPause()方法，然后调用 onStop()方法，最后调用 onDestroy()方法，结束当前 Activity。

Activity 生命周期可分为可视生命周期和活动生命周期。可视生命周期是 Activity

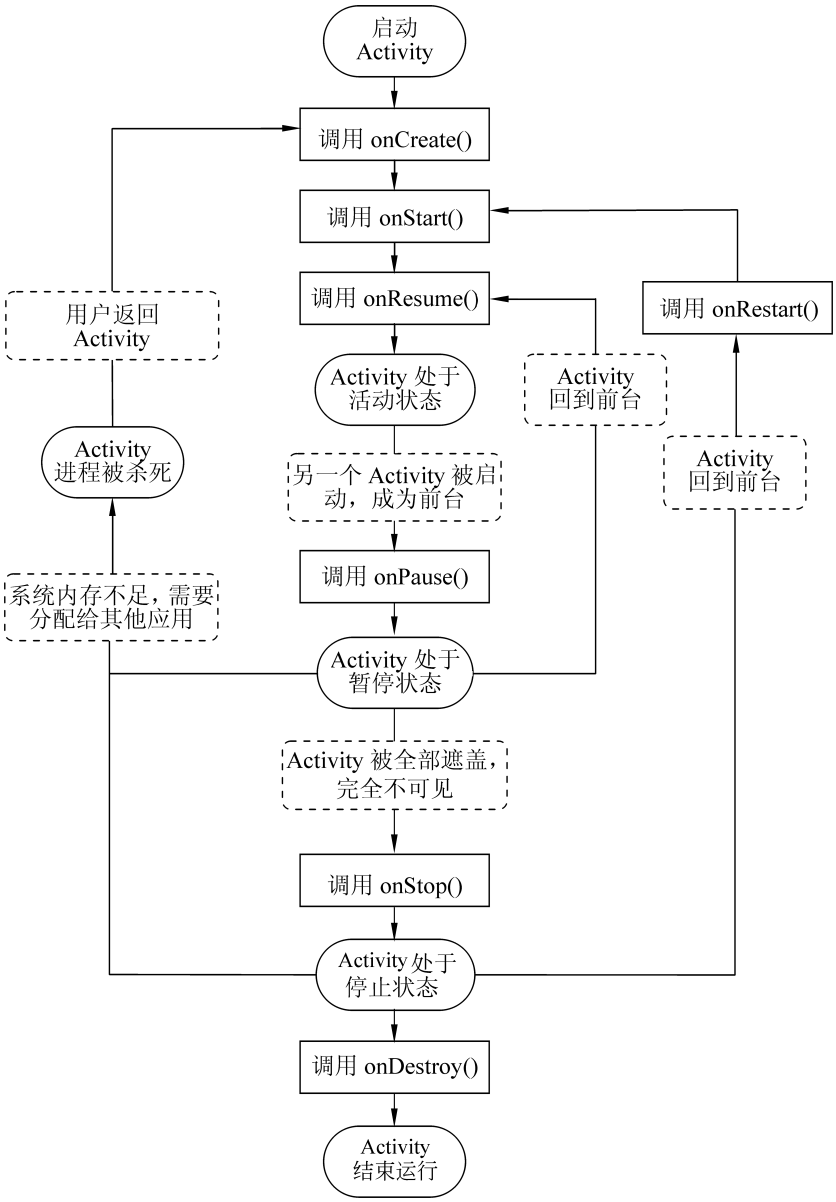

图 3-1　Activity 生命周期

在界面上从可见到不可见的过程，开始于 onStart()，结束于 onStop()。活动生命周期是 Activity 在屏幕的最上层，并能够与用户交互的阶段，开始于 onResume()，结束于 onPause()。在 Activity 的状态变换过程中，onResume()和 onPause()经常被调用，因此这两个方法中应使用简单、高效的代码。

编程人员可以通过重写生命周期方法在 Activity 中定义当处于什么状态时做什么事情。例如，当第一次启动一个 Activity 时，会调用 onCreate()方法，则初始化的操作可以写在这个方法中。

【例 3-1】　示例工程 Demo_03_ActivityLifeCycle 用于验证 Activity 生命周期方法被调用的情况，其主要代码如代码段 3-3 所示。


Activity 的生命周期

代码段 3-3　验证 Activity 生命周期方法的示例程序

```
//package 和 import 语句省略
public class MainActivity extends AppCompatActivity {
    private static final String TAG = "生命周期示例";
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Log.d(TAG, "-- onCreate()被调用--");
    }
    @Override
    protected void onStart() {
        super.onStart();
        Log.d(TAG, "-- onStart()被调用--");
    }
    ⋮       //其余代码类似
}
```

运行这个 Activity 后，在 Logcat 面板中能看到给出的提示信息，从中可看到 Activity 的生命周期是如何运行的。例如，启动这个 Activity 后 Logcat 面板输出的提示信息如图 3-2 所示，onCreate()、onStart()和 onResume()方法被依次调用；关闭这个 Activity 时，Logcat 面板输出的提示信息如图 3-3 所示，onPause()、onStop()和 onDestroy()方法被依次调用。

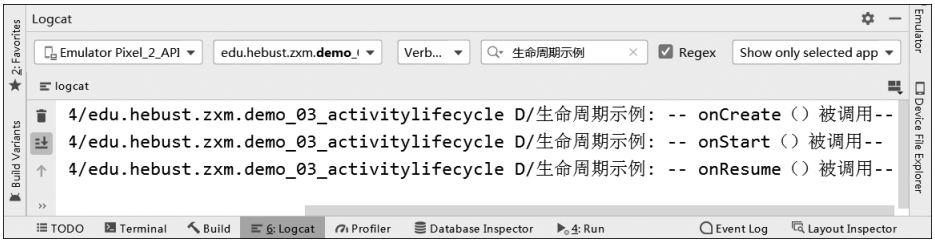

图 3-2　Activity 启动时调用的生命周期方法

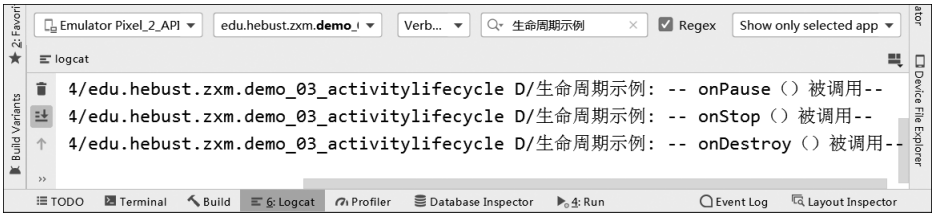

图 3-3　Activity 结束时调用的生命周期方法

3.1.4 Activity 的启动模式

主 Activity 在启动应用程序时就创建完毕,在其中可以显示 XML 布局信息、指定处理逻辑等。对于功能较复杂的应用程序,往往一个界面是不够用的,这就需要多个 Activity 来实现不同的用户界面。在 Android 系统中,所有的 Activity 由堆栈进行管理,Activity 栈遵循“后进先出”的规则。如图 3-4 所示,当一个新的 Activity 被执行后,它将会被放置到堆栈的最顶端,并且变成当前活动的 Activity,而先前的 Activity 原则上还是会存在于堆栈中,但它此时不会在前台。Android 系统会自动记录从首个 Activity 到其他 Activity 的所有跳转记录并且自动将以前的 Activity 压入系统堆栈,用户可以通过编程的方式删除历史堆栈中的 Activity 实例。

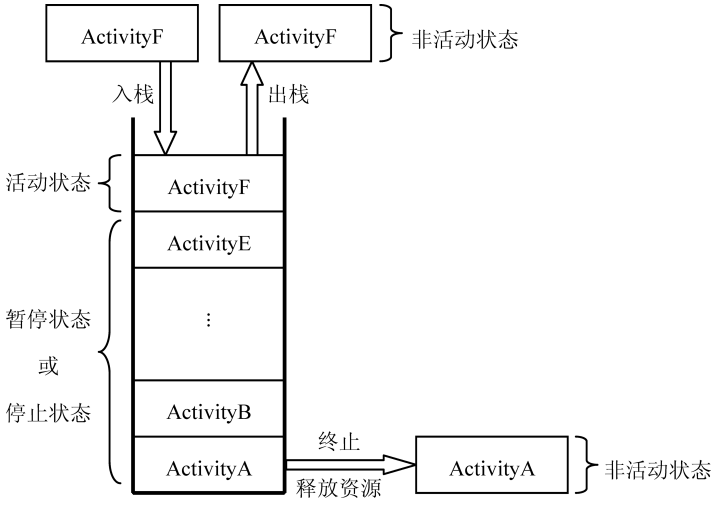

图 3-4 Activity 栈示意图

Activity 启动模式有 4 种,分别是 standard、singleTop、singleTask、singleInstance。可根据实际需求为 Activity 设置对应的启动模式,从而避免创建大量重复的 Activity 等问题。设置 Activity 启动模式的方法是在 AndroidManifest.xml 里对应的<activity>标签中设置 android:launchMode 属性,如图 3-5 所示。

图 3-5 在 AndroidManifest.XML 中设置 Activity 的启动模式

(1) standard 是默认模式,如果 Activity 已经启动,再次启动会创建一个新的

Activity 实例叠在前一个 Activity 之上。此时 Activity 是在同一个任务栈里，只不过是不同的实例。如果点击手机上的回退键会按照栈顺序依次退出。

（2）singleTop 模式是可以有多个实例的，但是不允许多个相同的 Activity 叠加在一起，如果 Activity 在栈顶时启动相同的 Activity，则不会创建新的实例。

（3）singleTask 模式中每个 Activity 只有一个实例。在同一个应用程序中启动 Activity 时，若它不存在，则会在当前创建一个新的实例；若存在，则会把任务列表中在其之上的其他 Activity 取消并调用它的 onNewIntent()方法。

（4）singleInstance 模式只有一个实例，不允许有别的 Activity 存在，也就是说，一个实例栈中只有一个 Activity。

3.1.5　Context 及其在 Activity 中的应用

Context 的中文解释是“上下文”或“环境”，在 Android 中应该理解为“场景”。例如，正在打电话时，场景就是用户所能看到的在手机里显示出来的拨号键盘，以及虽然看不到，但是却在系统后台运行的对应着拨号功能的处理程序。

Context 描述的是一个应用程序环境的上下文信息，是访问全局信息（如字符串资源、图片资源等）的接口。通过它可以获取应用程序的资源和类，也包括一些应用级别操作，如启动 Activity、启动和停止 Service、发送广播、接收 Intent 信息等。也就是说，如果需要访问全局信息，就要使用 Context。在代码段 3-4 中，this 指的是这个语句所在的 Activity 对象，同时也是这个 Activity 的 Context。

代码段 3-4　通过 Context 获取 Activity 上下文中的字符串信息

```
public class MainActivity extends AppCompatActivity {
    private TextView tv;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        tv = new TextView(this);          //通过 this 得到当前 Activity 的上下文信息
        tv.setText(R.string.hello_world);    //通过 Context 得到字符串资源
        setContentView(tv);
    }
}
```

Android 系统中有很多 Context 对象。例如，前述的 Activity 继承自 Context，也就是说每个 Activity 对应一个 Context；Service 也继承自 Context，每个 Service 也对应一个 Context。

常用的 Context 对象有两种：一种是 Activity 的 Context；另一种是 Application 的 Context。二者的生命周期不同：Activity 的 Context 生命周期仅在 Activity 存在时，也就是说，如果 Activity 已经被系统回收了，那么对应的 Context 也就不存在了；而 Application 的 Context 生命周期却很长，只要应用程序运行着，这个 Context 就是存在的，所以要根据自己程序的需要使用合适的 Context。

3.2 布局及其加载

Activity 主要用于呈现用户界面,包括显示 UI 控件、监听并处理用户界面事件并做出响应等。从本节开始,将系统地介绍 Android 的界面布局及其加载与控制。

3.2.1 View 类和 ViewGroup 类

在一个 Android 应用程序中,用户界面一般由一组 View 和 ViewGroup 对象组成。

View 对象是继承自 View 基类的可视化控件对象,是 Android 平台上表示用户界面的基本单元,如 TextView、Button、CheckBox 等。View 是所有可视化控件的基类,提供了控件绘制和事件处理的属性和基本方法,任何继承自 View 的子类都会拥有 View 类的属性及方法。表 3-1 给出了 View 类的部分常用属性的说明。View 及其子类的相关属性既可以在 XML 布局文件中进行设置,也可以通过成员方法在 Java 代码中动态设置。

表 3-1 View 类的部分常用属性

XML 属性	在 Java 代码中对应的方法	功能及使用说明
android:background	setBackgroundResource (int)	设置背景颜色
android:clickable	setClickable (boolean)	设置是否响应点击事件
android:focusable	setFocusable (boolean)	设置 View 控件是否能捕获焦点
android:id	setId(int)	设置 View 控件标识符
android:layout_width	setWidth(int)	设置宽度
android:layout_height	setHeight(int)	设置高度
android:text	setText(CharSequence)/ setText(int)	设置控件上显示的文字
android:textSize	setTextSize (float)	设置控件上显示文字的大小
android:textColor	setTextColor(int)	设置控件上显示文字的颜色
android:visibility	setVisibility (int)	设置 View 控件是否可见

ViewGroup 类是 View 类的子类,与 View 类不同的是,它可以充当其他控件的容器。ViewGroup 类作为一个基类为布局提供服务,其主要功能是装载和管理一组 View 和其他 ViewGroup,可以嵌套 ViewGroup 和 View 对象。Android 用户界面中的控件都是按照层次树的结构堆叠的,其关系如图 3-6 所示,而它们共同组建的顶层视图可以由应用程序中的 Activity 调用 setContentView()方法来显示。Android 中的一些复杂控件(如 Galley、GridView 等)都继承自 ViewGroup。

一般很少直接用 View 和 ViewGroup 来设计界面布局,更多的时候是使用它们的子类控件或容器来构建布局。常见用布局和 UI 控件的继承结构如图 3-7 所示。每个 View 和 ViewGroup 对象都支持它们自己的各种属性。一些属性对所有 View 对象可用,因为它们是从 View 基类继承而来的,如 id 属性;而有些属性只有特定的某一种 View 对象和它们的子类可用,如 TextView 及其子类支持 textSize 属性。

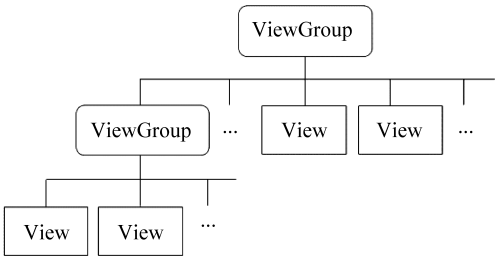

图 3-6　View 与 ViewGroup 的关系

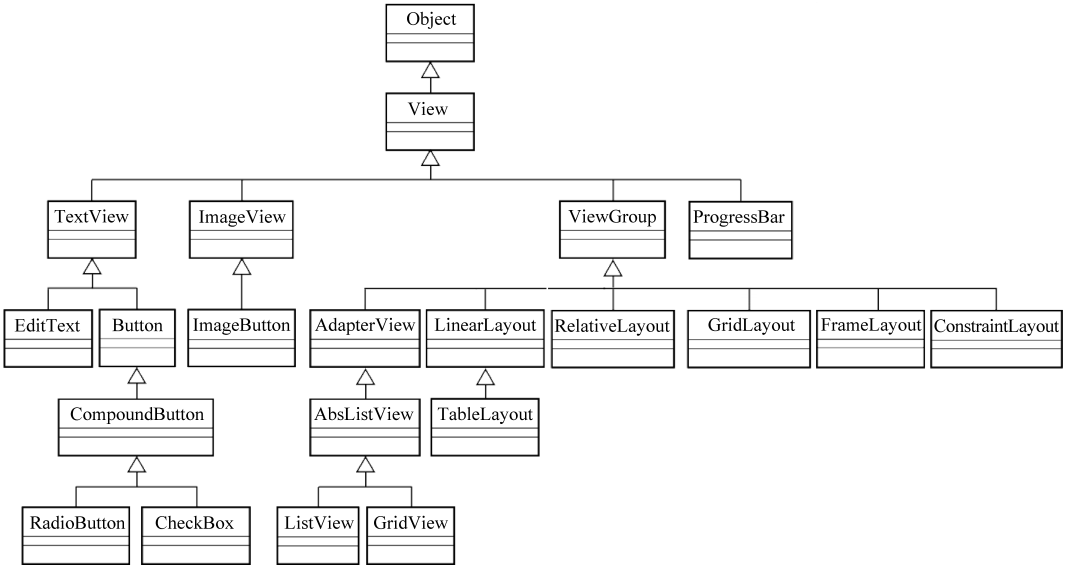

图 3-7　常用布局和 UI 控件的继承结构

3.2.2　XML 布局及其加载

在 Android 应用程序中，常见的布局方式有线性布局（LinearLayout）、相对布局（RelativeLayout）、表格布局（TableLayout）、网格布局（GridLayout）、帧布局（FrameLayout）、约束布局（ConstraintLayout）等。这些布局都通过 ViewGroup 的子类实现。

界面的布局可以在 XML 文件中进行设置，也可以通过 Java 代码设计实现。如果采用第一种方式，则需要在资源文件夹 res\layout 中定义相应的布局文件。这个 XML 布局文件由许多 View 对象嵌套组成。如果布局中有多个元素，那么最顶层的根节点必须是 ViewGroup 对象；如果整个布局只有一个元素，那么最顶层元素就是唯一的元素，它可以是一个单一的 UI 对象。

代码段 3-5 是一个自定义的布局文件 mylayout.xml，在其中声明了布局的实例，该例采用线性布局，布局中包括一个 TextView 控件。

代码段 3-5　自定义的 XML 布局文件

```
<?xml version="1.0" encoding="utf-8"?>
<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/tvHello"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="@string/hello"
        />
</LinearLayout>
```

定义了布局文件之后,需要在 Activity 中的 onCreate()回调方法中通过调用 setContentView()方法来加载这个布局,如代码段 3-7 所示。

代码段 3-6　通过重写 onCreate()方法加载用户界面的布局

```
public class MainActivity extends Activity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.mylayout);          //加载布局
    }
}
```

3.2.3　在 Activity 中定义和加载布局

除了上述直接调用已经设定好的 XML 布局外,还可以在 Java 代码中直接定义并加载某种布局,此时就不需要在工程的 res 文件夹下存放 XML 布局文件了。在将 UI 对象实例化并设置属性值后,通过调用 addView()方法可将其添加到设定的布局。

【例 3-2】　示例工程 Demo_03_DefineLayoutInActivity 演示在 Java 代码中定义并引用布局的方法。

此例是通过在 MainActivity 中添加线性布局而非通过 XML 布局文件来设置布局的。通过循环语句定义了 3 个按钮,并通过 addView()方法将其添加到布局中,如代码段 3-7 所示。

代码段 3-7　在 Activity 中设定布局

```
public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        LinearLayout myLayout=new LinearLayout(this);
                                                    //通过上下文设定线性布局对象
```

```
        myLayout.setGravity(Gravity.CENTER_HORIZONTAL);
        myLayout.setOrientation(LinearLayout.VERTICAL); //垂直布局
        myLayout.setPadding(0,20,0,0);                  //设置左、上、右、下边距
        setContentView(myLayout);                       //加载布局
        Button myBtn;                                   //定义按钮对象
        for (int i=1; i<4; i++){                        //添加几个按钮
            myBtn=new Button(this);
            myBtn.setText("按钮" + i);
            myBtn.setTextSize(20);                      //设置字体大小
            myBtn.setHeight(ViewGroup.LayoutParams.WRAP_CONTENT);
                                                        //设置按钮高度
            myLayout.addView(myBtn);                    //添加到布局中
            myBtn.getLayoutParams().width=500;          //设置按钮宽度
        }
    }
}
```

示例工程的运行结果如图 3-8 所示。

图 3-8　示例工程的运行结果

3.2.4　资源的管理与使用

在 Android 系统中，对字符、颜色、图像、音频、视频等资源的使用与管理也是很方便的，只要引用或设置资源文件夹 res 下的相关媒体文件或 XML 文件，就可以实现相关功能。

Android 应用程序中，XML 布局和资源文件并不包含在 Activity 的 Java 源码中，各种资源文件由系统自动生成的 R 文件来管理。每个资源类型在 R 文件中都有一个对应的内部类。例如，类型为 layout 的资源项在 R 文件中对应的内部类是 layout，而类型为 string 的资源项在 R 文件中对应的内部类就是 string。R 文件的作用相当于一个项目字典，项目中的用户界面、字符串、图片、声音等资源都会在对应的内部类中创建其唯一的 id，当项目中使用这些资源时，会通过该 id 得到资源的引用。如果程序开发人员变更了任何资源文件的内容或属性，R 文件会随之变动并自动更新其相应的内部类，开发者不需要也不能修改此文件。

在 Java 程序中通过 R 文件引用资源的方法是“R.资源类型.资源名称”，其中，资源类型可以是图像、字符串或布局文件，资源名称是资源文件名或 XML 文件中的对象名。例

如：R.drawable.background 表示引用资源文件夹中的 res\drawable\background.png 图片文件；R.string.title 表示引用资源文件 res\values\string.xml 中定义的 title 字符串；R.layout.activity_main 表示引用资源文件夹中的 res\layout\activity_main.xml 布局文件；R.id.tv_result 表示引用布局文件中 id 为 tv_result 的 TextView 对象。

1. 图片资源的管理与使用

Android Studio 工程项目提供了 mipmap 文件夹和 drawable 文件夹管理图片资源文件。新建工程项目时，系统会在资源文件夹 res 中自动创建多个 drawable 或 mipmap 文件夹，如 drawable-hdpi、drawable-mdpi、mipmap-hdpi、mipmap-mdpi 等，具体取决于 Android Studio 的版本。当应用程序安装在不同显示分辨率的终端上时，程序会自适应地选择加载某个文件夹中的资源。例如，一部屏幕密度为 320 的手机，会自动使用 drawable_xhdpi 文件夹下的图片。如果有默认文件夹 drawable，则系统在其他 dpi 文件夹下找不到图片时会使用 drawable 中的图片。

谷歌公司建议将应用程序的图标文件放在 mipmap 文件夹中，这样可以提高系统渲染图片的速度，提高图片质量，减小 GPU 压力。mipmap 支持多尺度缩放，系统会根据当前缩放范围选择 mipmap 文件夹中适当的图片，而不是像 drawable 文件夹根据当前设备的屏幕密度选择恰当的图片。

【例 3-3】 示例工程 Demo_03_UseImageResource 以设置 ImageView 的图片属性为例演示了如何在 XML 文件中引用图片资源。

首先将图片文件复制到工程中的 mipmap 文件夹下，图 3-9 是把 background.jpg 复制到工程中的效果。

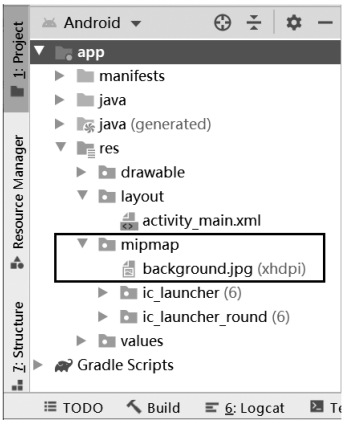

图 3-9 在工程中添加图片

在布局 XML 文件中，通过“@mipmap/图片文件名”的方式引用 mipmap 文件夹中的图片文件，实现代码如代码段 3-8 所示，运行结果如图 3-10 所示。

代码段 3-8 在 XML 文件中引用图片资源

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

```
<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:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="在 ImageView 中引用图片资源:"
        android:textColor="#000000"
        android:textSize="25sp"
        android:layout_margin="16dp"/>
     <ImageView
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         android:src="@mipmap/background"
         android:scaleType="centerCrop"/>
</LinearLayout>
```

图 3-10　设置 ImageView 的图片属性

如果是在 Java 代码中，则通过“R.mipmap.图片文件名”的方式引用 mipmap 文件夹中的图片文件。例如代码段 3-9 将 mipmap 文件夹中 background.jpg 设置为 App 的背景。

代码段 3-9　在 Activity 中设定 App 的背景

```
protected void onCreate(Bundle savedInstanceState) {
```

```
        super.onCreate(savedInstanceState);
        this.getWindow().setBackgroundDrawableResource(R.mipmap.background);
        //用指定图片作为背景
        setContentView(R.layout.activity_main);
    }
```

2. 字符串资源的管理与使用

字符串资源描述文件 strings.xml 一般位于工程 res 文件夹下的 values 子文件夹中。如果需要在 Activity 代码或布局文件中使用字符串,可以在 strings.xml 文件中的<resources>标签下添加相应的<string>元素,定义字符串资源。<string>元素的基本格式如下:

```
<string name="字符串名称"> 字符串的内容 </string>
```

代码段 3-10 是一个典型的 strings.xml 示例,其中定义了两个字符串资源。

代码段 3-10　string.xml 代码示例

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">MyFirstApplication</string>
    <string name="hello_world">Hello world!</string>
</resources>
```

上述代码段的第 1 行定义了 XML 版本与编码方式;第 2 行以后在<resources>标签下定义了两个<string>元素,定义了两个字符串,字符串的名称分别为 app_name 和 hello_world。如果需要在 Java 程序代码中使用这些字符串,可以用“R.string.字符串名称”的方式引用。如果在 XML 文件中使用这些字符串,则用“@string/字符串名称”的方式引用。Android 解析器会从工程的 res/values/strings.xml 文件里读取相应名称对应的字符串值并进行替换。

3. 数组资源的管理与使用

与字符串资源类似,数组描述文件 arrays.xml 位于工程 res 文件夹下的 values 子文件夹中。数组资源也定义在<resources>标签下,其基本语法如下:

```
<数据类型-array name="数组名">
    <item>数组元素值 1</item>
    <item>数组元素值 2</item>
    ⋮
</数据类型-array>
```

代码段 3-11 是一个典型的 arrays.xml 示例,在其中定义了两个字符串数组,数组名

分别是 citys 和 modes。

代码段 3-11　arrays.xml 代码示例

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string-array name="citys">
        <item>北京</item>
        <item>天津</item>
        <item>上海</item>
    </string-array>
    <integer-array name="modes">
        <item>1</item>
        <item>2</item>
        <item>3</item>
    </integer-array>
</resources>
```

在 XML 中引用数组资源的方法是“@array/数组名称”，例如：

```
<Spinner
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:entries="@array/citys"/>
```

在 Java 代码中引用数组资源的方法是“getResources().getXxxArray(R.array.数组名称)”，例如：

```
String[] citys=getResources().getStringArray(R.array.citys);
int[] modes=getResources().getIntArray(R.array.modes);
```

4. 颜色描述资源的管理与使用

颜色描述文件 colors.xml 位于工程 res 文件夹下的 values 子文件夹中，其典型内容如代码段 3-12 所示。

代码段 3-12　colors.xml 代码示例

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="colorPrimary">#3F51B5</color>
    <color name="colorPrimaryDark">#303F9F</color>
    <color name="colorAccent">#FF4081</color>
</resources>
```

在＜resources＞标签下添加相应的＜color＞元素，定义颜色资源，其基本格式如下：

```
<color name="颜色名称"> 该颜色的值 </color>
```

颜色值通常为 8 位的十六进制的颜色值,表达式顺序是 # aarrggbb,其中 aa 表示 alpha 值(00 为完全透明,FF 为完全不透明),aa 可省略,此时表示一个完全不透明的颜色;rr 表示红色分量值;gg 表示绿色分量值;bb 表示蓝色分量值。例如,#7F0400FF 表示半透明的蓝色。任何一种颜色的值范围都是十六进制 00～FF(0～255)。

在 XML 中引用颜色资源的方法是“@color/颜色名称”,例如:

```
android:textColor="@color/colorAccent"
```

在 Java 代码中引用颜色资源的方法是“getResources().getColor(R.color.颜色名称)”,或“ContextCompat.getColor(context, R.color.颜色名称)”,例如:

```
TextView hello=(TextView)findViewById(R.id.hello);
hello.setTextColor(getResources().getColor(R.color.colorPrimary));
```

5. 引用 assets 文件夹中的资源

同 res 文件夹相似,assets 也是存放资源文件的文件夹,但 res 文件夹中的内容会被编译器所编译,assets 文件夹则不会。也就是说,应用程序运行的时候,res 文件夹中的内容会在启动的时候载入内存,assets 文件夹中的内容只有在被用到的时候才会载入内存,所以一般将一些不经常用到的大资源文件存放在该文件夹下,如应用程序中用到的音频、视频、图片、文本等文件。

在程序中可以使用“getResources.getAssets().open("文件名")”的方式得到资源文件的输入流 InputStream 对象。

3.3 常用的布局

3.3.1 线性布局 LinearLayout

线性布局将其包含的子元素按水平或者垂直方向顺序排列。布局方向由属性 android:orientation 的值来决定,其值为 vertical 时子元素垂直排列,为 horizontal 时子元素水平排列。同时,可使用 android:gravity 属性调整其子元素向左、向右或居中对齐,或使用 android:padding 属性来微调各子元素的摆放位置,还可以通过设置子元素的 android:layout_weight 属性值控制各个元素在容器中的相对大小。

在 XML 布局文件中,线性布局的子元素定义在<LinearLayout></LinearLayout>标签之间。每个线性布局的所有子元素,如果垂直分布则仅占一列,如果水平分布则仅占一行。线性布局中如果子元素所需位置超过一行或一列,不会自动换行或换列,超出屏幕的子元素将不会被显示。

【例 3-4】　示例工程 Demo_03_LinearLayout 演示了线性布局的用法。

在 Android Studio 中新建一个工程，选用空白的 Activity 模板，系统会自动为该 Activity 建立一个位于 res/layout/中的布局文件，自动建立的内容采用约束布局。可以把这个约束布局直接修改为线性布局，如代码段 3-13 所示。本例中采用垂直布局，在布局中添加了 3 个按钮，示例工程运行结果如图 3-11 所示。

图 3-11　线性布局示例工程的运行结果

代码段 3-13　线性布局示例

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:gravity="center_horizontal"
    android:paddingTop="8dp">
    <Button
        android:text="按钮 1"
        android:id="@+id/button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
    <Button
        android:text="按钮 2"
        android:id="@+id/button2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
    <Button
        android:text="按钮 3"
        android:id="@+id/button3"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
</LinearLayout>
```

在代码段 3-13 中，为每个按钮对象定义了 id 属性。在 XML 布局文件中，可以通过设置 android:id 属性来给相应的 UI 元素指定 id，通常是以字符串的形式定义一个 id，格

式如下:

```
android:id="@+id/id 字符串"
```

这里在“@+id/”后面的字符是设定的 id,@表示 XML 解析器应该解析 id 字符串并把它作为 id 资源;+表示这是一个新的资源名字,它被创建后应加入 R 资源文件中。通过这个 id,可在 XML 布局或 Activity 代码中引用相应的 UI 元素。引用 id 时不需要符号+。例如,代码段 3-14 演示了在 Activity 中通过 id 引用布局中的第一个按钮,并设置其 text 属性值。

代码段 3-14 通过 findViewById()方法引用布局中的 UI 对象

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    Button myButton=(Button)findViewById(R.id.button1);
    //取得 Button 控件句柄,存储到 myButton 对象中
    myButton.setText("hello");
    //字符串 hello 显示在 id 为 button1 的按钮上面
}
```

另外,一般情况下每个 View 对象都需要设置宽度和高度(layout_width 和 layout_height)属性。可以指定宽度和高度的绝对值,如 50dp,也可指定为 match_parent 或者 wrap_content。match_parent 使 UI 元素对象扩展以填充布局单元内尽可能多的空间,如果设置一个元素的 layout_width 和 layout_height 属性值为 match_parent,它将被强制性布满整个父容器。而 wrap_content 使 UI 元素对象扩展以显示其全部内容,例如,TextView 和 ImageView 对象,设置其 layout_width 和 layout_height 属性值为 wrap_content,将恰好完整显示其内部的文本或图像,UI 元素将会根据其内容自动更改大小并包裹住文字或图片内容。除此之外,还可以通过设置 UI 元素对象的对齐方式、边距、边界等,调整其在界面中的位置。例如,代码段 3-13 中,通过设置根布局的 android:gravity 属性使 3 个按钮水平居中。

线性布局
LinearLayout

【例 3-5】 示例工程 Demo_03_BrowserByLinearLayout 采用线性布局,实现了一个简易浏览器界面。

本例演示了线性布局中子元素在容器中的相对大小比例的控制。本例使用了 android:layout_weight 属性,该属性用于定义控件对象所占空间分割父容器的比例。

android:layout_weight 属性只有在 LinearLayout 中才有效,其默认值为 0。其含义是一旦 View 对象设置了该属性,那么该对象的所占空间等于 android:layout_width 或 layout_height 设置的空间加上剩余空间的占比。即 LinearLayout 如果显式包含 layout_weight 属性,则会计算两次对象所占尺寸,第一次将正常计算所有 View 对象的宽高,第二次将结合 layout_weight 的值分配剩余的空间。例如,假设屏幕宽度为 L,两个 View 对象的宽度都为 match_parent,其 layout_weight 的值分别是 1 和 2,则两个 View 的原有

宽度都为 L，那么剩余宽度为 $L-(L+L)=-L$，第一个 View 对象占比为 1/3，所以总宽度是 $L+(-L)\times 1/3=(2/3)L$。

一般推荐当使用 layout_weight 属性时，将 android:layout_width 或 layout_height 设为 0dp，这样 layout_weight 值就可以简单理解为空间占比了。

示例工程的布局效果预览如图 3-12(a)所示，运行结果如图 3-12(b)所示。XML 布局文件的内容如代码段 3-15 所示。

(a) 布局效果预览　　(b) 运行结果

图 3-12　线性布局实现简易浏览器界面

代码段 3-15　采用线性布局实现简易浏览器界面

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="8dp">
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="horizontal">
        <EditText
            android:id="@+id/editText"
```