华清远见 FARSIGHT 始于2004 | 嵌入式培训专家

10年口碑积累，成功培养50000多名研发工程师，铸就专业品牌形象

华清远见的企业理念是不仅要做良心教育、做专业教育，更要做受人尊敬的职业教育。

# 《Android 应用程序开发与典型案例》

作者：华清远见

专业始于专注 卓识源于远见

## 第 3 章 程序设计基础

本章简介

在上一章的学习中，主要了解了 Eclipse+ADT 的开发流程，对其有了初步的认识和了解。对初学者来说，这一章的内容比较烦琐，但是又必须掌握，这也是进行 Android 开发必须经过的第一步，有了这个基础，我们将进行 Android 应用程序设计。

专业始于专注 卓识源于远见

# 3.1 Android 程序框架

上一章我们建立了 AndroidTest 项目，在项目中，所有的代码是由 ADT 插件自动生成的，我们并没有对其进行编码，所以没有对其框架进行分析。其实每一个平台都有自己的结构框架，比如，在最初学习 Java 或者 C/C++时，第一个程序总是 main 方法，以及文件类型和存储方式等。本节将对 Android 平台的目录结构、文件类型及其负责的功能和 Android 平台的 main 方法进行剖析。

## 3.1.1 Android 项目目录结构

有了前两章的基础，再来打开上一章建立的 AndroidTest 项目，分析其项目目录结构，对 Android 项目进一步深入了解。首先启动 Eclipse，展开“Package Explorer”导航器中的“AndroidTest”项目，如图 3-1 所示。

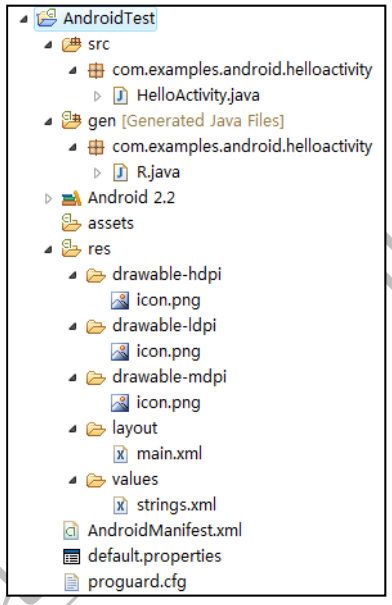


图 3-1 JDK 安装检查 Android 项目结构

与一般的 Java 项目一样，src 文件夹是项目的所有包及源文件（.java），res 文件夹中则包含了项目中的所有资源，比如，程序图标（drawable）、布局文件（layout）、常量（values）等。下面来介绍其他 Java 项目中没有的 gen 文件夹中的 R.java 文件和每个 Android 项目都必须有的 AndroidManfest.xml 文件。

R.java 是在建立项目时自动生成的，这个文件是只读模式，不能更改，R.java 文件是定义该项目所有资源的索引文件。先来看看 AndroidTest 项目的 R.java 文件，如代码清单 3-1 所示。

代码清单 3-1 R.java

```
package com.examples.android.helloactivity;

public final class R{
 public static final class attr{
}
public static final class drawable{
 public static final int icon=0x7f020000;
}
public static final class layout{
 public static final int main=0x7f030000;
}
public static final class string{
 public static final int app_name=0x7f040001;
 public static final int hello=0x7f040000;
}
}
```

可以看到这里定义了很多常量，仔细一看就发现这些常量的名字都与 res 文件夹中的文件名相同，再次证明 R.java 文件中所存储的是该项目所有资源的索引。有了这个文件，在程序中使用资源将变得更加方便，可以很快地找到要使用的资源，由于这个文件不能被手动编辑，所以当在项目中加入了新的资源时，只需要刷新一下该项目，R.java 文件便自动生成了所有资源的索引。

AndroidManfest.xml 文件则包含了该项目中所使用的 Activity、Service、Receiver，先来打开 AndroidTest 项目中的 AndroidManfest.xml 文件，如代码清单 3-2 所示。

代码清单 3-2　AndroidManfest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
package=" com.examples.android.helloactivity"
   android:versionCode="1"
android:versionName="1.0" >

<application android:icon="@drawable/icon"
                            android:label="@string/app_name">
      <activity android:name=".HelloActivity"
              android:label="@string/app_name">
         <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category
                  android:name="android.intent.category.LAUNCHER" />
         </intent-filter>
      </activity>
   </application>
</manifest>
```

代码清单 3-2 中，intent-filters 描述了 Activity 启动的位置和时间。每当一个 Activity（或操作系统）要执行一个操作时，它将创建出一个 Intent 的对象，这个 Intent 对象能承载的信息可描述你想做什么，你想处理什么数据，数据的类型，以及一些其他信息。而 Android 则会和每个 Application 所暴露的 intent-filter 的数据进行比较，找到最合适 Activity 来处理调用者所指定的数据和操作。下面我们来仔细分析 AndroidManfest.xml 文件，如表 3-1 所示。

表 3-1　AndroidManfest.xml 分析

| manifest | 根节点，描述了 package 中所有的内容 |
|---|---|
| xmlns:android | 包含命名空间的声明。xmlns:android=http://schemas.android.com/apk/res/android，使得 Android 中各种标准属性能在文件中使用，提供了大部分元素中的数据 |
| package | 声明应用程序包 |
| application | 包含 Package 中 Application 级别组件声明的根节点。此元素也可包含 Application 的一些全局和默认的属性，如标签、icon、主题、必要的权限，等等。一个 manifest 能包含零个或一个此元素（不能大于一个） |
| android:icon | 应用程序图标 |
| android:label | 应用程序名字 |
| activity | 用来与用户交互的主要工具。Activity 是用户打开一个应用程序的初始页面，大部分被使用到的其他页面也由不同的 Activity 所实现，并声明在另外的 Activity 标记中。注意，每一个 Activity 必须有一个<Activity>标记对应，无论它给外部使用或是只用于自己的 Package 中。如果一个 Activity 没有对应的标记，你将不能运行它。另外，为了支持运行时查找 Activity，可包含一个或多个<intent-filter>元素来描述 Activity 所支持的操作 |
| android:name | 应用程序默认启动的 Activity |
| intent-filter | 声明了指定的一组组件支持的 Intent 值，从而形成了 IntentFilter。除了能在此元素下指定不同类型的值，属性也能放在这里来描述一个操作所需的唯一的标签、icon 和其他信息 |
| action | 组件支持的 Intent action |

| | |
|---|---|
| category | 组件支持的 Intent Category。这里指定了应用程序默认启动的 Activity |
| uses-sdk | 该应用程序所使用的 SDK 版本相关 |

下面我们看看资源文件中一些常量的定义，如 String.xml，如代码清单 3-3 所示。

代码清单 3-3　String.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
 <string name="hello">Hello World, HelloActivity!</string>
 <string name="app_name">HelloWorld</string>
</resources>
```

这个文件很简单，就定义了两个字符串资源，与 R.java 中对应的索引如代码清单 3-4 所示。

代码清单 3-4　R.java 中的 String 类

```
public static final class string{
 public static final int app_name=0x7f040001;
 public static final int hello=0x7f040000;
}
```

在程序中装载并使用这个字符串资源如代码清单 3-5 所示。

代码清单 3-5　String 资源的使用

```
Resources r = this.getContext().getResources();
String appname = ((String) r.getString(R.string.appname));
String hello = ((String) r.getString(R.string.hello));
```

基本上可以定义出项目中所有使用的常量，例如颜色。所以，可根据需要对资源常量进行定义。下面是定义了颜色的常量 colors.xml，如代码清单 3-6 所示。

代码清单 3-6　colors.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
 <color name="status_idle">#cccccc</color>
 <color name="status_done">#637a47</color>
<color name="status_sync">#cc9900</color>
<color name="status_error">#ac4444</color>
</resources>
```

现在我们来分析 AndroidTest 项目的布局文件（layout），首先打开 res→layout→main.xml 文件，如代码清单 3-7 所示。

代码清单 3-7　main.xml

```
<?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"
   >
   <TextView
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:text="@string/hello"
   />
</LinearLayout>
```

在代码清单 3-7 中，有以下几个布局和参数。

- ＜LinearLayout>：线性版面配置，在这个标签中，所有元件都是由上到下排成的。
- android:orientation：表示这个介质的版面配置方式是从上到下垂直地排列其内部的视图。这里“vertical”表示是水平排列。

- android:layout_width：定义当前视图在屏幕上所占的宽度，fill_parent/ match_parent 即填充整个屏幕。
- android:layout_height：定义当前视图在屏幕上所占的高度，fill_parent/ match_parent 即填充整个屏幕。
- wrap_content：随着文字大小的不同而改变这个视图的宽度或高度。
- layout_weight ：用于给一个线性布局中的多个视图的重要度赋值。所有视图都有 layout_weight 值，默认为零，即需要显示多大的视图就占据多大的屏幕空间。如果值大于零，则将父视图中的可用空间分割，分割大小具体取决于每一个视图的 layout_weight 值和该值在当前屏幕布局的整体 layout_weight 值，以及在其他视图屏幕布局的 layout_weight 值中所占的比例。

在这里，布局中设置了一个 TextView，用来配置文本标签 Widget，其中设置的属性 android:layout_width 为整个屏幕的宽度，android:layout_height 可以根据文字来改变高度，而 android:text 则设置了这个 TextView 要显示的文字内容，这里引用了@string 中的 hello 字符串，即 String.xml 文件中的 hello 所代表的字符串资源。hello 字符串的内容"Hello World, HelloActivity!"这就是在 AndroidTest 项目运行时看到的字符串。

最后，分析 AndroidTest 项目的主程序文件 HelloActivity.java，如代码清单 3-8 所示。

代码清单 3-8　HelloActivity.java

```
package com.examples.android.helloactivity;

import android.app.Activity;
import android.os.Bundle;

public class HelloActivity extends Activity
{
    public void onCreate(Bundle savedInstanceState)
    {
       super.onCreate(savedInstanceState);
       setContentView(R.layout.main);
    }
}
```

主程序 HelloActivity 类继承自 Activity 类，重写了 void onCreate（Bundle savedInstance State）方法。在 onCreate 方法中通过 setContentView（R.layout.main）设置了 Activity 要显示的布局文件（/layout/main.xml）。

到这里，是不是明白了为什么在创建项目时没有进行编码就可以直接运行程序呢？当然，这也是 Android 开发的特点，这样可以很轻松地将代码和 UI 分开，在国际化和程序维护方面有着巨大的作用。如果 Android 程序需要自适应国际化，比如说多国语言等问题，那么就可以定义不同语言的 UI 布局，在程序装载时调用不同的布局。而且，如果需要修改 UI 的一些问题，就不必查看代码了，直接更改这些布局文件即可，是不是很方便？当然，这需要开发者在开发时使用这种 MVC 框架，尽量减少使用“硬编码”。

### 3.1.2　Android 应用解析

上面了解了 Android 应用程序的目录结构和其中每个文件的功能，要对其进行应用开发，还需要对 Android 应用构造进行深入的分析。下面介绍一下 Android 应用开发中所应用到的重要模块。

- Activity
- Intent
- Content Provider
- Service

当然，并非所有的 Android 应用程序都必须由这 4 部分组成，它可以根据开发者需求来进行组合，比如，上面建立的 HelloActivity 项目就只使用了 Activity 这一个模块。但是，对于任何一个应用程序来说，都必须在 AndroidManfest.xml 文件中声明使用到的模块。

1．Activity

Android 中最基本的模块就是 Activity，在之前的 HelloActivity 项目中已经使用过。我们称之为“活动”，在应用程序中，一个活动（Activity）通常就是一个单独的屏幕。每一个活动都被实现为一个独立的类，并且继承于从活动基类，活动类将会显示由视图控件组成的用户接口，并对事件作出响应。例如，HelloActivity 项目中的 HelloActivity.java 即继承了活动（Activity）类。大多数的应用都是由多个 Activity 显示组成，例如，对一个文本信息应用而言，第一个屏幕用来显示发送消息的联系人列表，第二个屏幕用来写文本消息和选择收件人，第三个屏幕查看消息历史或者消息设置操作等。

这里的每一个屏幕就是一个活动，很容易实现从一个屏幕到一个新的屏幕，并且完成新的活动。当一个新的屏幕打开后，前一个屏幕将会暂停，并保存在历史栈中。用户可以返回到历史栈中的前一个屏幕，当屏幕不再使用时，还可以从历史栈中删除。

简单理解，Activity 就代表着用户所能看到的屏幕，它主要处理了应用程序的整体性工作，例如，为用户显示指定的 View，启动其他 Activity，监听系统事件（按键事件、触摸屏事件等）等。

所有应用的 Activity 都继承于 Android 提供的基类 android.app.Activity 类，其他的 Activity 继承该父类后，通过父类的方法来实现各种功能，这种设计在其他领域也较为常见。

### 2．Intent

在 Android 中，利用 Intent 类实现了在 Activity1 与 Activity2 之间的切换（以及启动 Service 等）。Intent 类主要用于描述应用的功能。在 Intent 的描述结构中，有动作（Action）和动作对应的数据（data）两个最重要的部分。其中，典型的动作类型有：MAIN、VIEW、PICK、EDIT 等，而动作对应的数据以 URI 的形式表示。例如，如果要查看一个人的联系方式，需要先创建一个动作类型为 VIEW 的 Intent，以及一个表示这个动作对应的数据（这里就是这个人）的 URI。

从一个屏幕导航到另一个屏幕，我们需要解析各种 Intent。为了实现向前导航，首先，我们调用 Activity 的 startActivity（Intent myIntent）方法。此时，系统为找到最匹配 myIntent 的 Intent 对应的 Activity，就将在所有已安装的应用程序中定义的 IntentFilter 中查找。而匹配的对应的新的 Activity 在接收到 myIntent 的通知后，开始运行。当 startActivity 方法被调用时，将触发解析 myIntent 的动作，该机制提供了两个关键好处。

（1）Activities 能够重复利用从其他组件中以 Intent 的形式产生的请求。

（2）Activities 可以在任何时候被具有相同 Action 的新的 Activity 取代。

下面举例说明两个 Activity 之间的切换。运行效果：当应用程序启动时，显示布局 main.xml，如图 3-2 所示，当我们单击“切换”按钮时，屏幕显示布局 main2.xml，如图 3-3 所示，再单击“切换”按钮，又回到如图 3-2 所示的状态。就这样通过 Intent 完成了两个 Activity 之间的切换。

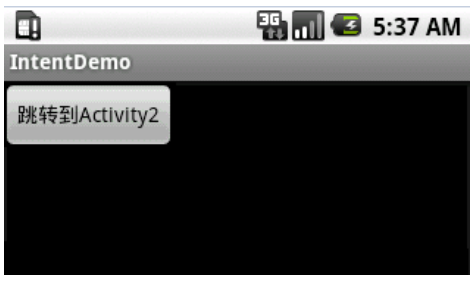


图 3-2　Activity1

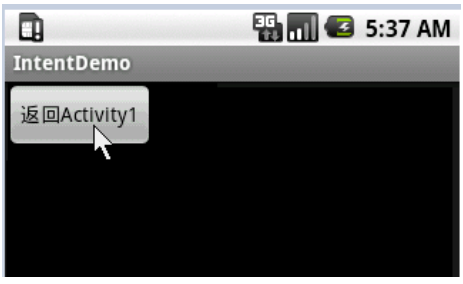


图 3-3　Activity2

以下是两个 Activity 的代码。

代码清单 3-9　Activity1.java

```
package com.example.android.Examples_03_01;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
```

```
/ **
 * 在 Examples_02_01 项目中一共使用了两个 Activity,
* 每使用一个 Activity，都必须在 AndroidManifest.xml 中
* 进行声明
*/
public class Activity1 extends Activity
{
     public void onCreate(Bundle savedInstanceState)
     {
         super.onCreate(savedInstanceState);
         /* 设置显示 main.xml 布局 */
         setContentView(R.layout.main);
         /* findViewById(R.id.button1)取得布局 main.xml 中的 button1 */
         Button button = (Button) findViewById(R.id.button1);
         /* 监听 button 的事件信息 */
         button.setOnClickListener(new Button.OnClickListener() {
             public void onClick(View v)
             {
                 /* 新建一个 Intent 对象 */
                 Intent intent = new Intent();
                 /* 指定 intent 要启动的类 */
                 intent.setClass(Activity1.this, Activity2.class);
                 /* 启动一个新的 Activity */
                 startActivity(intent);
                 /* 关闭当前的 Activity */
                 Activity1.this.finish();
             }
         });
     }
}
```

代码清单 3-10 Activity02.java

```
package com.example.android.Examples_03_01;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;

public class Activity2 extends Activity
{
     public void onCreate(Bundle savedInstanceState)
     {
         super.onCreate(savedInstanceState);
         /* 设置显示 main2.xml 布局 */
         setContentView(R.layout.main2);
         /* findViewById(R.id.button2)取得布局 main.xml 中的 button2 */
         Button button = (Button) findViewById(R.id.button2);
         /* 监听 button 的事件信息 */
         button.setOnClickListener(new Button.OnClickListener() {
             public void onClick(View v)
             {
                 /* 新建一个 Intent 对象 */
                 Intent intent = new Intent();
                 /* 指定 intent 要启动的类 */
                 intent.setClass(Activity2.this, Activity1.class);
                 /* 启动一个新的 Activity */
                 startActivity(intent);
                 /* 关闭当前的 Activity */
                 Activity2.this.finish();
             }
         });
     }
}
```

代码清单 3-11 main.xml

```
<?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"
    >

    <Button
      android:id="@+id/button1"
      android:layout_width="100px"
      android:layout_height="wrap_content"
      android:layout_x="100px"
      android:layout_y="80px"
      android:text="跳转到 Activity2"
    >
    </Button>
</LinearLayout>
```

代码清单 3-12 Main2.xml

```
<?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"
    >

    </TextView>
    <Button
      android:id="@+id/button2"
      android:layout_width="100px"
      android:layout_height="wrap_content"
      android:layout_x="100px"
      android:layout_y="80px"
      android:text="返回 Activity1"
    >
    </Button>
</LinearLayout>
```

如代码清单 3-9 所示，需要在 AndroidManifest.xml 中声明使用的 Activity2，如代码清单 3-13 所示。

代码清单 3-13 AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
     package="com.example.android.Examples_03_01"
     android:versionCode="1"
     android:versionName="1.0">
    <application android:icon="@drawable/icon" android:label="@string/app_name">
       <activity android:name=".Activity1"
                android:label="@string/app_name">
           <intent-filter>
              <action android:name="android.intent.action.MAIN" />
              <category android:name="android.intent.category.LAUNCHER" />
           </intent-filter>
       </activity>
       <activity android:name="Activity2"></activity>
    </application>
    <uses-sdk android:minSdkVersion="8" />
</manifest>
```

问：Android 应用如何对外部事件（如当电话呼入时，或者数据网络可用时，或者到了晚上时）做出响应？

答：使用 IntentReceiver。

- 在相应的事件发生时，IntentReceiver 使用 NotificationManager 通知用户，但它并不能生成 UI。
- 它在 AndroidManifest.xml 中注册，也可以在代码中使用 Context.registerReceiver()进行注册。
- 当 IntentReceiver 被触发时，应用不必对请求调用 IntentReceiver，系统会在需要时启动应用。
- 各种应用还可以通过使用 Context.broadcastIntent()将它们自己的 intentreceiver 广播给其他应用。

### 3．Content Provider

Android 应用能够将它们的数据保存到文件和 SQLite 数据库中，甚至是任何有效的设备中。利用 Content Provider 实现应用数据与其他的应用共享。因为 Content Provider 类实现了一组标准的方法，能够让其他的应用保存或读取此内容提供器处理的各种数据类型。

Content Provider 可以理解为在不同的应用包之间共享数据的工具。在 Android 中，默认的系统数据库为 SQLite。但是在 Android 中，使用方法略有区别。在 Android 中，每一个应用都单独运行在各自的进程中，当一个应用需要访问其他应用的数据时，这里就需要在不同的虚拟机之间传递数据，这样的情况操作起来可能有些困难（正常情况下，你不能读取其他应用的 db 文件）。Content Provider 就解决了上述困难。

在 Android 中，Content Povider 是一个特殊的存储数据的类型，它提供了一套标准的接口用来获取和操作数据。并且，Android 自身也提供了现成的 Content Provider：Contacts、Browser、CallLog、Settings、MediaStore。当通过 Content Resolver 提供的方法来使用 Content Provider 时，应用可以通过唯一的 Content Resolver interface 来使用具体的某个 Content Provider。其中，Content Resolver 提供的方法包括 query()、insert()、update()等。要使用这些方法，还会涉及 URI。可以将它理解成 string 形式的 Content Provider 的完全路径。

### 4．Service

Service 即“服务”的意思，既然是服务，那么 Service 将是一个生命周期长而且没有用户界面的程序。比如，一个正在从播放列表中播放歌曲的媒体播放器，在这个媒体播放器应用中，应该会有多个 Activity，让使用者可以选择歌曲并播放歌曲。然而，音乐重放这个功能并没有对应的 Activity，因为使用者会认为在导航到其他屏幕时音乐应该还在播放。在这个例子中，媒体播放器这个 Activity 会使用 Context.Startservice()来启动一个 Service，从而可以在后台保持音乐的播放。同时，系统也将保持这个 Service 一直执行，直到这个 Service 运行结束。另外，还可以通过使用 Context.Bindservice()方法连接到一个 Service 上（如果这个 Service 当前还没有处于启动状态，则将启动它）。当连接到一个 Service 之后，还可用 Service 提供的接口与它进行通信。以媒体播放器为例，还可以执行暂停、重播等操作。

## 3.2 Android 程序 UI 设计

在前面章节的例子中，我们已经接触了 TextView、Button 等 UI 控件，其实这里所说的 UI 就是布局文件（layout），UI 是一个应用程序展现给用户的界面，一个应用程序要想受用户喜爱，那么 UI 可不能差。自从 Android SDK 1.0_r2 版本开始，ADT 提供了 UI 预览的功能。现在只需要打开一个 Android 项目的“/res/layout/main.xml”并右键单击鼠标右键，在弹出的快捷菜单中选择“Open With”→ “Android Layout Editor”命令，或者直接双击 main.xml 文件，即可以切换到 UI 设计界面，如图 3-4 所示。

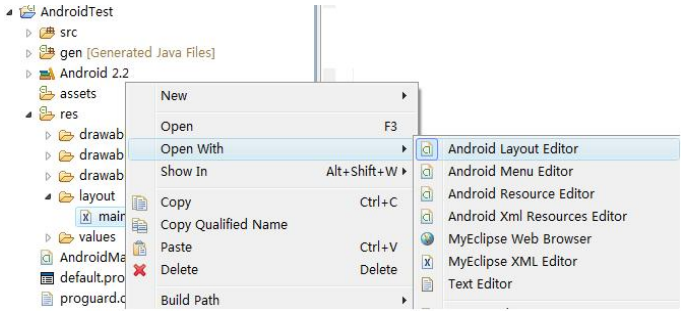

图 3-4　Android Layout Editor 命令

左边的 layouts 标签的内容则是一些线性布局，可以使用它轻松地完成对布局的排版，比如，横向或者纵向布局。Views 标签则是一些 UI 控件，可以将这些控件直接拖动到右边的窗口进行编辑，这些 UI 控件的类型如图 3-5 所示。

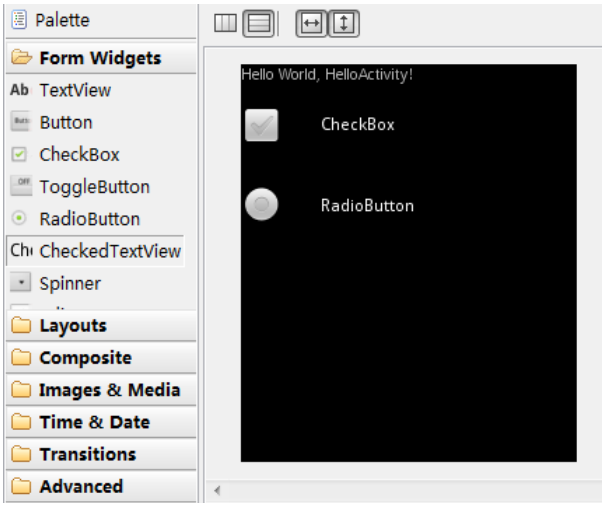


图 3-5　Android layout Editor

当然，还可以单击右下角的 main.xml 标签来切换到 XML 编辑器，对代码进行编排，如图 3-6 所示。将这些功能配合起来使用，基本可以满足开发者需求。

```
<?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"
    >
<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="@string/hello"
    />
<CheckBox
    android:text="CheckBox"
    android:id="@+id/checkBox1"
```

图 3-6　XML 编辑器

## 3.3　Java 语言在 Android 程序中的使用

Android 应用程序采用 Java 语言编写，Java 语法和 C/C++有很大的相似性，但也有一些特别之处。

### 3.3.1　Interface 的使用

从名字上看，Interface 即为接口的意思，多用于实现回调（Call Back）方法。

在 Interface 的定义中，一般的代码架构如代码清单 3-14 所示。

代码清单 3-14　InterfaceServer.java

```
public class InterfaceServer {
  public interface OnClickListener{
       public void onClick();
  }
  private OnClickListener mOnClickListener=null;
```

```
    public void onClick(){
         if(mOnClickListener!=null)
              mOnClickListener.onClick();
    }
    public void setOnClickListener(OnClickListener l){
         mOnClickListener = l;
    }
```

对于 Interface 内部的方法而言，只需要声明，而不需要具体实现。从编译器的角度来看，Interface 会被认为是一个指向方法的指针。

使用 InterfaceServer 的代码一般如代码清单 3-15 所示。

代码清单 3-15　使用 InterfaceServer

```
public void addToButton {
  Button b = (Button)findViewById(R.id.button);
  onClickListener l = new OnClickListener(){
       public void onClick(View v){
       TextView tv1 = (TextView) findViewById(R.id.tv1);
       tv1.setText("The Button has been clicked");
  }
};
b.setOnClickListener(l);
}
```

### 3.3.2　abstract class 的使用

abstract 是一个修饰符，其类似于 Static 这样的关键字。Android 程序中常用 abstract 修饰一个类，如 abstract class，当然能它也可以修饰一些变量或是方法。

抽象类所包含的方法可以只是定义，也可以是已实现的。对于没有实现的方法，基于该方法的子类必须实现；而对于已经实现的方法，子类可以重写该方法，若没有重写，则使用父类的方法。

在一定程度上，abstract class 可以代替 Interface，例如，3.3.1 节中 Interface 的例子做如下的 abstract class 替换，其效果是等价的。

代码清单 3-16　InterfaceServer.java

```
public class InterfaceServer {
  abstract class OnClickListener2{
       public void onClick2();
  }
  private OnClickListener2 mOnClickListener2=null;
  public void onClick2(){
       if(mOnClickListener2!=null)
            mOnClickListener2.onClick2();
  }
  public void setOnClickListener2(OnClickListener2 l){
       mOnClickListener2 = l;
  }
```

### 3.3.3　Interface 与 abstract class 的区别

从语法角度讲，接口和抽象类有以下区别。

- Java 语法规定，一个子类只能有一个父类，但可以实现多个接口。
- abstract class 可以代替 Interface。
- 定义 Interface 时，只需要列出所包含方法的定义而不必实现。而定义 Abstract 类时，方法必须有实现部分，这就是所谓的默认实现，除非该方法也是 Abstract 类型。
- 接口的子类必须实现接口所定义的全部方法，而抽象类的子类不必实现抽象类所定义的任何方法，除非该方法是 Abstract 或者子类想重写某个方法。

- 接口中的成员变量必须是 Static Final 类型（实际应用中则很少包含变量，因为接口多用于引用），而 abstract class 内部可以包含任意变量。

从应用的角度来讲，Interface 和 abstract class 的区别在于：Interface 提供了一个方法集合的接口，该接口用于客户端和服务端的方法调用，如图 3-7 所示。

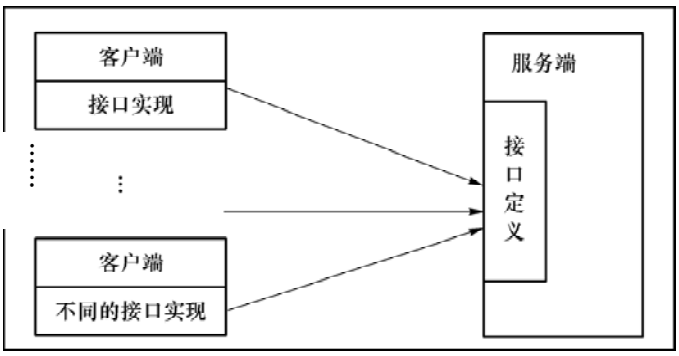


图 3-7　Interface 的使用机制

接口一般是由服务端定义，比如操作系统，客户端根据自己的需求对接口做不同的实现；而 abstract class 则仅提供了一个基类，该基类没有任何服务端或者客户端的概念，它的作用就是为了继承并重写，如图 3-8 所示。

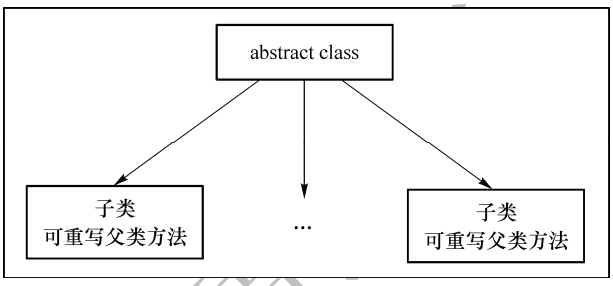


图 3-8　abstract class 的使用机制

### 3.3.4　for 循环的使用

除了传统的 for 循环（语法是 for(inti=0;i<N;i++){};）以外还有 foreach 循环。举例如代码清单 3-17 所示。

代码清单 3-17　For 循环语法

```
public void doSomething(){
     int[] ages = new int[20];
     for(int age:ages){
          //to do,add something to process age.
          Log.i("Haiii" . "The age is" +age);
          …
     }
}
```

该 for 语法是对某个集合进行循环，第 1 个参数是循环过程集合元素值的引用，第 2 个参数是集合对象。第 1 个参数的类型必须和集合元素的类型相同。

以下代码示例了传统的 for 语法和新 for 语法的等效使用，但由于编译器对新 for 语法的优化，其执行效率将更高。

代码清单 3-18　传统 for 语法和新 for 语法比较

```
public void doSomething(){
     int[] counts =new int[20];
     int total1=0,total2=0;
     for(int i=0;i<20;i++)
          total1 += counts[i];
     for(int count:counts)
```

```
            total2 += count;
    }
```

### 3.3.5 Map 类的使用

在 Android 系统中，有着多种存储数据的方式，例如，文件、数据库及程序内参数式存储、网络存储等。对于参数式存储时，使用的就是 Map 类。Map 本身是 Interface，Java 基于该接口实现三个具体的 Map 类，分别是 HashMap、TreeMap，以及 EnumMap，常用的为 HashMap，本节也主要介绍 HashMap。

Map 定义了访问特定集合的标准方法，这种集合用来存储 key-value 类型的键值对，比如，对于 name:Haiii 和 age:22 这两组数据来讲，其中 name、age 称为键（key），与此对应的是键值（value）。在一个 Map 集合类中,每对键或值其类型都可以是任意的,比如 int、String 等都是可以的。

Map 类又是一个类模板，一个 Map 类对象在初始化时必须指定键的类型，可以是任何 Object 类，比如，Map<String,Object> mMap= new HashMap<String,Object>()。

<>里面的数据类型用于指定 Map 集合中“键值对”的类型。

给 Map 集合添加和删除键值对的方法如表 3-2 所示。

**表 3-2 Map 集合添加和删除键值对的方法**

| 方法 | 描述 |
|---|---|
| clear() | 删除该 Map 集合中的全部元素 |
| remove(Objectkey) | 删除键名为 key 所对应的键值对 |
| put(Objectkey,Objectvalue) | 添加一个新的键值对 |
| putAll(Mapmap) | 将该 Map 集合的元素全部复制到新的 Map 中 |

Map 类没有提供直接遍历键值对的方法，要遍历所有键值对需要一个中间过程。Map 提供了 3 个方法用于间接遍历键值对，如下：

- entrySet() 返回所有键值对类型为 Set 对象。
- keySet() 返回所有键值对类型为 Set 对象。
- valueSet() 返回所有键值对类型为 Collection 对象。

要得到具体的键值对，需要再解析 Set 和 Collection 对象，但仅有这两个对象还不能获得键值对，还需要借助于 Iterator 类。到这里，可能觉得有些复杂，别着急，结果马上就要出来了。

Set、Collection、Iterator 实际上是 Map 内部进行操作的 3 个辅助类，要得到具体 Map 键值对，如代码清单 3-19 所示。

**代码清单 3-19 得到具体的 Map 键值对**

```
Map<String,Object> mMap = new HashMap<String,Object>();

Iterator kv = mMap.entrySet().iterator();
Iterator k = mMap.keySet().iterator();
Iterator v = mMap.values().iterator();

Int size = mMap.size();
for(int i = 0;i<size;i++)
{
 Map.Entryentry = (Map.Entry)kv.next();
 Object key = entry.getKey();
 Object value = entry.getValue();
}
```

用以上代码读取键值对时，Object 可以强制转换为 int 类型。

### 3.3.6 Integer 与 String 之间的转换

在实际程序设计中，经常需要把 Integer 类型转换为 String 类型，或者相反，Java 类库中提供了这样的方法。

- 整形转换为字符串：String.valueOf(Stringstr)。
- 字符串转换为整形：Integer.parseInt(int)、Interger.parseLong(long)等。

面向对象编程中，一般不能直接调用类中的方法，而是需要先定义类的一个对象，然后才能使用其包含的方法。而以上两个方法直接调用 String 和 Integer 的方法，这就是 static 关键字的作用。

在定义一个类时,如果其中的方法声明为 static，那么外部程序就可以直接调用该方法，该方法所引用的一些变量也必须为 static 类型的变量。

Java 内部有一种安全机制：对于一个普通的类，必须声明该类的对象才能访问类中的方法或者变量，实现这种安全机制的是 Java 编译器。编译器隐藏了所有类的地址，因此不能通过类名称定位到类的地址；而如果使用 static 修饰符，无论是变量还是方法,编译器都会把该变量或者方法的名称导出，以便程序能够根据类名定位到类所在的地址，从而能够调用相应的方法或者变量。

### 3.3.7 synchronized 同步关键字

synchronized 关键字属于操作系统的范畴，与同步对应的是异步。在程序设计的概念中,有同步调用或者异步调用，同步是指该段代码（方法）从调用开始,直到内部执行完毕后才能返回；异步调用是指调用该段代码（方法）后立即返回，无论该段代码内部所执行的物理操作是否执行完毕，异步代码一般存在于多线程程序设计中，单线程程序内部不存在异步调用。

synchronized 关键字的作用就是告诉操作系统，在执行该关键字所限定的代码片段内，不允许被其他线程打断。在一般的操作系统设计中，会提供一个类似于 synchronized 的 API 方法，而 Java 则是给了这样一个关键字，相当于说，Java 编译器为操作系统分担了一部分工作。

那么，什么情况下需要使用 synchronized 关键字呢？凡是需要某段代码在执行时不被其他线程打断时，都可以加上 synchronized 关键字，举例如代码清单 3-20 所示。

代码清单 3-20 synchronized 关键字

```
public class MyMusicWidgetProvider{
 private static MyMusicWidgetProvider sInstance;
 static synchronized MyMusicWidgetProvider getInstance(){
      if(sInstance==null)
          sInstance=new MyMusicWidgetProvider();
      returnsInstance;
 }
}
```

以上代码定义了一个类，并希望该类在运行时仅有一个实例，每次调用该类所包含的方法时，先得到该类的实例，然后再通过实例调用其他方法。该类有一个 static 的 getInstance()方法，该方法的作用就是检查是否存在一个实例，如果没有就创建一个,否则返回存在的实例。

getInstance()方法前面加了 synchronized 关键字，为的是该方法在执行时不能被其他线程打断，那么，为什么有这样的要求呢？试想一下，假设没有使用该关键字,第 1 个线程 A 在执行 getInstance()方法时，方法体中的 new 运算符刚刚创建了一个 MyMusicWidgetProvider 实例，但还没来得及把实例赋值给 sInstance，而此时 B 线程又来调用 getInstance()方法，出现这种情况的原因是：sInstance= new MyMusicWidgetProvider()这句代码会被 Java 编译器编译成多条机器指令，其中给 sInstance 赋值的机器指令和创建一个 MyMusicWidgetProvider 对象的机器指令是分开的。此时对于 B 线程来讲，它会重新检查 sInstance 是否为空，由于 A 线程还没有来得及给 sIntance 赋值，因此，B 线程就会再次创建一个新的 MyMusicWidgetProvider 实例，当 B 线程暂停并返回 A 线程时，A 线程会把第 1 次创建的实例重新赋值给 sInstance，这就会导致 B 线程将来对 MyMusicWidgetProvider 实例的错误引用，因为 B 线程所创建实例的地址被 A 线程修改了。

因此，这就要求 getInstance()一旦开始执行，必须执行完毕后才能返回，中间不能被其他线程打断。

sychronized 除了约束整个方法外,也可以约束一小段代码，这样做的好处有两个：一个是可以避免定义新方法；另一个是能够避免一些线程同步问题，如代码清单 3-21 所示。

代码清单 3-21 synchronized 约束小段代码

```
Object obj;
```

```
int c;
void addMethod(){
  synchronized(obj){
       c++;
       //其他代码
  }
}
```

obj 可以是任意类型的一个对象，该对象的唯一作用就是给 synchronized 提供一个“锁”。同步设计的原则之一是尽量减少同步中包含的代码大小，从而在线程间能够更平衡地运行。因此，同步一小段代码是一个不错的做法。同时，假设 addMethod()内部还调用了其他同步方法，那么，从这个同步跳到另外的同步会增加线程间死锁的风险，因此不同步整个 addMethod()而仅同步局部代码,这就是第 2 个好处。

### 3.3.8 new 的使用

Java 语言中，new 的作用是为一个对象（Object）分配内存，代码清单 3-22 说明了为各种 Object 分配内存的方法。

代码清单 3-22 为各种 Object 分配内存的方法

```
int a = 20;
int A[]= new int[100];
float A2[]= new float[100];
int A3[]= {10,20,30};
String str = new String();
String str1 = ;
String str2 = null;
String[] Str = new String[100];
MyMusicWidgetProvider myProvider = new MyMusicWidgetProvider();
str1 += "Android is from... ";
MyMusicWidgetProvider commonProvider= myProvider.getInstance()
```

一般情况下，没有用 new 修饰符定义的数据都是在栈（Stack）中分配内存，但有一个例外，对于 String 定义的变量，总是从系统内存堆（Heap）中分配内存，栈中仅有对该 String 的引用。另外，从系统堆中分配的实际内存大小并不是按指定的大小分配的，比如，int A[]=new int[100]所分配的内存大小并不是 100B，而是 128B，内存分配机制为了提高分配效率以及分配算法的可实现性，实际上的内存颗粒大小会按照 2 的幂次方进行划分，实际分配的内存大小是最接近指定大小的一个值。另外，最小的内存颗粒大小会根据不同的内存分配算法有所不同，一般会取 512B 或者 1KB。

问：在 Android 上使用 Java 与在 PC 上使用 Java 的编程风格有何差别?

答：Android 是一种嵌入式系统，在追求程序效率、降低所需资源上是孜孜不倦的，程序员心中始终要有该意识。在 PC 上进行编程时，清晰完整的程序结构比运行效率显得更重要，因此，程序结构上经常呈现多层结构，比如 getter、setter 等；而在 Android 上编程时，尽管需要清晰的程序结构，但对于一些小的结构，则是效率重于结构。因此，能在一个函数或引用中实现的功能就要避免进行多层嵌套调用。

## 3.4 本章小结

本章主要介绍了 Android 应用程序框架、UI 设计，以及 Java 在 Android 程序中的使用。首先彻底地分析了上一章的 AndroidTest 项目，从其项目目录结构、文件功能等方面分析了 Android 应用程序的基本框架，其次逐一分析了 Android 应用程序的构成，并分别通过示例程序演示了其功能的运用。其次介绍了有关 UI 设计的工具，使得程序界面更加漂亮。最后介绍了在 Android 程序中 Java 语法的使用。

关键知识点测评

1．以下有关 Android 程序目录结构的说法，不正确一个是（　　）。

A．Android 程序中包含其他 Java 项目中没有的 gen 文件夹

B．AndroidManfest.xml 文件则包含了该项目中所使用的 Activity、Service、Receiver

C．程序的布局文件（layout）中配置了程序的线性版面，视图位置、高度等

D．R.java 文件可以更改，被编辑

2．以下有关 Android 应用的叙述，正确的一个是（　　）。

A．所有的 Android 应用程序都有 Activity、Intent、Content Provider、Service 4 个模块构造而成

B．一个活动（Activity）通常就是一个单独的屏幕，它不需要继承其他基类

C．Intent 可以实现 Activity 与 Activity 之间的切换件

D．Service 能够让其他的应用保存或读取此内容提供器处理的各种数据类型

3．以下有关 interface 与 abstract class 区别的描述，不正确的是（　　）。

A．定义 abstract 类时，所有方法不需要有实现部分

B．一个子类只能有一个父类，但可以实现多个接口

C．接口中的成员变量必须是 static final，而 Abstract class 内部可以包含任意变量

D．接口的子类必须实现接口所定义的全部方法，而抽象类的子类不必实现抽象类所定义的任何方法

## 联系方式

华清远见

集团官网：www.hqyj.com　　嵌入式学院：www.embedu.org　　移动互联网学院：www.3g-edu.org

企业学院：www.farsight.com.cn　　物联网学院：www.topsight.cn　　研发中心：dev.hqyj.com

集团总部地址：北京市海淀区西三旗悦秀路北京明园大学校内 华清远见教育集团

北京地址：北京市海淀区西三旗悦秀路北京明园大学校区，电话：010-82600386/5

上海地址：上海市徐汇区漕溪路 250 号银海大厦 11 层 B 区，电话：021-54485127

深圳地址：深圳市龙华新区人民北路美丽 AAA 大厦 15 层，电话：0755-22193762

成都地址：成都市武侯区科华北路 99 号科华大厦 6 层，电话：028-85405115

南京地址：南京市白下区汉中路 185 号鸿运大厦 10 层，电话：025-86551900

武汉地址：武汉市工程大学卓刀泉校区科技孵化器大楼 8 层，电话：027-87804688

西安地址：西安市高新区高新一路 12 号创业大厦 D3 楼 5 层，电话：029-68785218