献　　词

献给我桌上的唱片机。感谢你陪伴我完成本书。我保证，很快你就会有新唱针了。

——B.P.

献给我的爸爸David，他教我懂得辛苦工作的意义。献给我的妈妈Lisa，她一直督促我去做正确的事。

——C.S.

献给我的爸爸Dave Vadas，他激励并支持我投身计算机行业。献给我的妈妈Joan Vadas，在这么多年的浮浮沉沉里，她总能让我保持乐观。（她会给我支招：心情不好的时候就看集《黄金女郎》吧！）

——K.M.

1.4.1　视图层级结构

组件包含在视图（View）对象的层级结构中，这种结构又称作*视图层级结构*（view hierarchy）。图1-11展示了代码清单1-2所示的XML布局对应的视图层级结构。

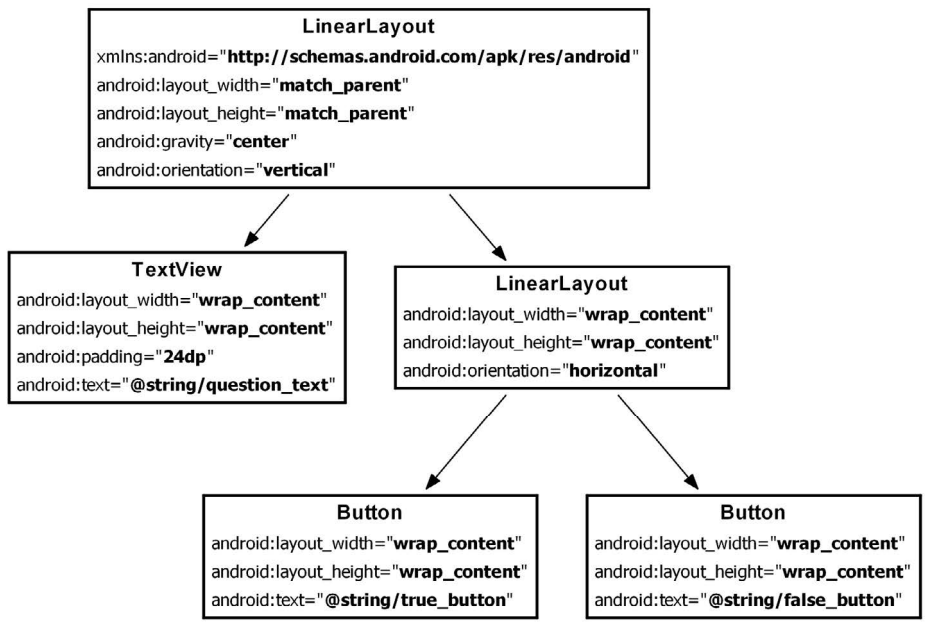

图1-11　布局组件的层级结构

从布局的视图层级结构可以看到，其根元素是一个LinearLayout组件。作为根元素，LinearLayout组件必须指定Android XML资源文件的命名空间属性，这里是http://schemas.android.com/apk/res/android。

LinearLayout组件继承自ViewGroup组件（也是个View子类）。ViewGroup组件是包含并配置其他组件的特殊组件。想要以一列或一排的样式布置组件，就可以使用LinearLayout组件。其他ViewGroup子类还有FrameLayout、TableLayout和RelativeLayout。

若某个组件包含在一个ViewGroup中，该组件与ViewGroup即构成父子关系。根LinearLayout有两个子组件：TextView和另一个LinearLayout。作为子组件的LinearLayout自己还有两个Button子组件。

1.4.2　组件属性

下面来看看配置组件时常用的一些属性。

1. android:layout_width和android:layout_height属性

几乎每类组件都需要android:layout_width和android:layout_height属性。以下是它

们的两个常见属性值（二选一）。

- ❑ match_parent：视图与其父视图大小相同。
- ❑ wrap_content：视图将根据其显示内容自动调整大小。

（以前还有个fill_parent属性值，等同于match_parent，现已废弃不用。）

根LinearLayout组件的高度与宽度属性值均为match_parent。LinearLayout虽然是根元素，但它也有父视图——Android提供该父视图来容纳应用的整个视图层级结构。

其他包含在界面布局中的组件，其高度与宽度属性值均被设置为wrap_content。请参照图1-10理解该属性值定义尺寸大小的作用。

TextView组件比其包含的文字内容区域稍大一些，这主要是android:padding="24dp"属性的作用。该属性告诉组件在决定大小时，除内容本身外，还需增加额外指定量的空间。这样屏幕上显示的问题与按钮之间便会留有一定的空间，使整体显得更为美观。（不理解dp的意思？dp即density-independent pixel，指与密度无关的像素，详见第9章。）

2. android:orientation属性

android:orientation属性是两个LinearLayout组件都具有的属性，它决定两者的子组件是水平放置还是垂直放置。根LinearLayout是垂直的，子LinearLayout是水平的。

子组件的定义顺序决定其在屏幕上显示的顺序。在垂直的LinearLayout中，第一个定义的子组件出现在屏幕的最上端；而在水平的LinearLayout中，第一个定义的子组件出现在屏幕的最左端。（如果设备文字从右至左显示，如阿拉伯语或者希伯来语，第一个定义的子组件则出现在屏幕的最右端。）

3. android:text属性

TextView与Button组件具有android:text属性。该属性指定组件要显示的文字内容。

请注意，android:text属性值不是字符串值，而是对字符串资源（string resource）的引用。

字符串资源包含在一个独立的名叫strings的XML文件中（strings.xml），虽然可以硬编码设置组件的文本属性值，如android:text="True"，但这通常不是个好主意。比较好的做法是：将文字内容放置在独立的字符串资源XML文件中，然后引用它们。这样会方便应用的本地化（支持多国语言）。

需要在activity_quiz.xml文件中引用的字符串资源还没添加。现在就来处理。

1.4.3 创建字符串资源

每个项目都包含一个默认字符串资源文件strings.xml。

在项目工具窗口中，找到app/res/values目录，展开目录，打开strings.xml文件。

可以看到，项目模板已经添加了一些字符串资源。如代码清单1-3所示，添加应用布局需要的三个新的字符串。

代码清单1-3 添加字符串资源（strings.xml）

```
<resources>
```

```
    <string name="app_name">GeoQuiz</string>
    <string name="question_text">Canberra is the capital of Australia.</string>
    <string name="true_button">True</string>
    <string name="false_button">False</string>
</resources>
```

（某些版本的Android Studio的strings.xml默认带有其他字符串，请勿删除它们，否则会引发与其他文件的联动错误。）

现在，在GeoQuiz项目的任何XML文件中，只要引用到`@string/false_button`，应用运行时，就会得到文本“False”。

保存strings.xml文件。这时，activity_quiz.xml布局缺少字符串资源的提示信息应该就消失了。（如仍有错误提示，请检查一下这两个文件，确认没有拼写错误。）

默认的字符串文件虽然已命名为strings.xml，你仍可以按个人喜好重命名。一个项目也可以有多个字符串文件。只要这些文件都放在res/values/目录下，含有一个`resources`根元素，以及多个`string`子元素，应用就能找到并正确使用它们。

1.4.4　预览布局

至此，应用的界面布局已经完成，可以使用图形布局工具实时预览了。首先，确认保存了所有相关文件并且无错误发生，然后回到activity_quiz.xml文件，点击代码编辑区右边的Preview打开预览工具窗口（如果还没打开的话），如图1-12所示。

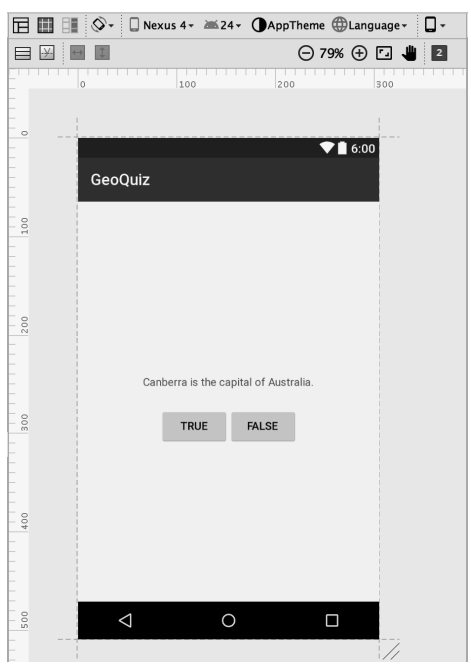

图1-12　在图形布局工具中预览布局（activity_quiz.xml）

1.5 从布局 XML 到视图对象

知道activity_quiz.xml中的XML元素是如何转换为视图对象的吗？答案就在于QuizActivity类。

在创建GeoQuiz项目的同时，向导也创建了一个名叫QuizActivity的Activity子类。QuizActivity类文件存放在项目的app/java目录下。java目录是项目全部Java源代码的存放处。

在项目工具窗口中，依次展开app/java目录与com.bignerdranch.android.geoquiz包。找到并打开QuizActivity.java文件，查看其中的代码，如代码清单1-4所示。

代码清单1-4 默认QuizActivity类文件（QuizActivity.java）

```
package com.bignerdranch.android.geoquiz;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;

public class QuizActivity extends AppCompatActivity {

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

（是不是不明白AppCompatActivity的作用？它实际就是一个Activity子类，能为Android旧版本系统提供兼容支持。第13章会详细介绍AppCompatActivity。）

如果无法看到全部类包导入语句，请单击第一行导入语句左边的⊕符号来显示它们。

该Java类文件有一个Activity方法：onCreate(Bundle)。

（如果你的文件还包含onCreateOptionsMenu(Menu)和onOptionsItemSelected(MenuItem)方法，暂时不用理会。第13章会详细介绍它们。）

activity子类的实例创建后，onCreate(Bundle)方法会被调用。activity创建后，它需要获取并管理用户界面。要获取activity的用户界面，可调用以下Activity方法：

```
public void setContentView(int layoutResID)
```

根据传入的布局资源ID参数，该方法生成指定布局的视图并将其放置在屏幕上。布局视图生成后，布局文件包含的组件也随之以各自的属性定义完成实例化。

资源与资源 ID

布局是一种资源。资源是应用非代码形式的内容，如图像文件、音频文件以及XML文件等。

项目的所有资源文件都存放在目录app/res的子目录下。在项目工具窗口中可以看到，activity_quiz.xml布局资源文件存放在res/layout/目录下。strings.xml字符串资源文件存放在res/values/目录下。

可以使用资源ID在代码中获取相应的资源。activity_quiz.xml布局的资源ID为R.layout.activity_quiz。

查看GeoQuiz应用的资源ID需要切换项目视图。Android Studio默认使用Android项目视图，如图1-13所示。为让开发者专注于最常用的文件和目录，默认视图隐藏了Android项目的真实文件目录结构。

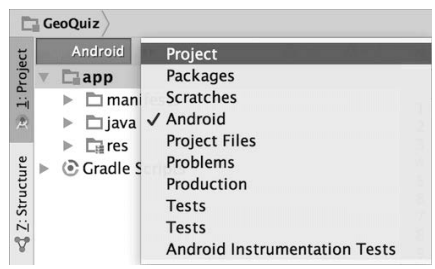

图1-13　切换项目视图

在项目工具窗口的最上部找到下拉菜单，从Android项目视图切换至Project视图。Project视图会显示出当前项目的所有文件和目录。

展开目录app/build/generated/source/r/debug，找到项目包名称并打开其中的R.java文件，即可看到GeoQuiz应用当前所有的资源。R.java文件在Android项目编译过程中自动生成，如该文件头部的警示所述，请不要修改该文件的内容。

修改布局或字符串等资源后，R.java文件不会实时刷新。Android Studio另外还存有一份代码编译用的R.java隐藏文件。当前代码编辑区打开的R.java文件仅在应用安装至设备或模拟器前产生，因此只有在Android Studio中点击运行应用时，它才会得到更新。

R.java文件通常比较大，代码清单1-5仅展示了部分内容。

代码清单1-5　GeoQuiz应用当前的资源ID（R.java）

```
/* AUTO-GENERATED FILE. DO NOT MODIFY.
 *
 * This class was automatically generated by the
 * aapt tool from the resource data it found. It
 * should not be modified by hand.
 */

package com.bignerdranch.android.geoquiz;

public final class R {
    public static final class anim {
        ...
    }

    ...

    public static final class id {
```

```
        ...
    }
    public static final class layout {
        ...
        public static final int activity_quiz=0x7f030017;
    }
    public static final class mipmap {
        public static final int ic_launcher=0x7f030000;
    }
    public static final class string {
        ...
        public static final int app_name=0x7f0a0010;
        public static final int false_button=0x7f0a0012;
        public static final int question_text=0x7f0a0014;
        public static final int true_button=0x7f0a0015;
    }
}
```

可以看到R.layout.activity_quiz即来自该文件。`activity_quiz`是R的内部类`layout`里的一个整型常量名。

GeoQuiz应用需要的字符串同样具有资源ID。目前为止，我们还未在代码中引用过字符串，如果需要，可以使用以下方法：

```
setTitle(R.string.app_name);
```

Android为整个布局文件以及各个字符串生成资源ID，但activity_quiz.xml布局文件中的组件除外，因为不是所有组件都需要资源ID。在本章中，我们要在代码里与两个按钮交互，因此只需为它们生成资源ID即可。

本书主要使用Android项目视图，生成资源ID之前，记得切回。当然，如果你就喜欢使用Project视图，也没啥问题。

要为组件生成资源ID，请在定义组件时为其添加`android:id`属性。在activity_quiz.xml文件中，分别为两个按钮添加上`android:id`属性，如代码清单1-6所示。

代码清单1-6 为按钮添加资源ID（activity_quiz.xml）

```
<LinearLayout  ... >

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:padding="24dp"
        android:text="@string/question_text" />

    <LinearLayout
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:orientation="horizontal">

        <Button
            android:id="@+id/true_button"
```

```
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/true_button" />

        <Button
            android:id="@+id/false_button"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/false_button" />

    </LinearLayout>

</LinearLayout>
```

注意，android:id属性值前面有一个+标志，而android:text属性值则没有。这是因为我们在创建资源ID，而对字符串资源只是做引用。

1.6　组件的实际应用

按钮有了资源ID，就可以在QuizActivity中直接获取它们。首先，在QuizActivity.java文件中增加两个成员变量。

在QuizActivity.java文件中输入代码清单1-7所示代码。（请勿使用代码自动补全功能。）

代码清单1-7　添加成员变量（QuizActivity.java）

```
public class QuizActivity extends AppCompatActivity {

    private Button mTrueButton;
    private Button mFalseButton;

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

文件保存后，会看到两个错误提示。没关系，稍后会处理。请注意新增的两个成员（实例）变量名称的m前缀。该前缀是Android编程应遵循的命名约定，本书将始终遵循该约定。

现在，将鼠标移至代码左边的错误提示处时，会看到两条同样的错误信息：Cannot resolve symbol 'Button'。

这告诉我们，需要在QuizActivity.java文件中导入android.widget.Button类包。可在文件头部手动输入以下代码：

```
import android.widget.Button;
```

或者使用Option+Return（Alt+Enter）组合键，让Android Studio自动为你导入。代码有误时，也可以使用该组合键来修正。记得要常用。

类包导入后，刚才的错误提示应该就消失了。（如果仍然存在，请检查Java代码以及XML文件，确认是否存在输入或拼写错误。）

接下来，我们来编码使用按钮组件，这需要以下两个步骤：

- 引用生成的视图对象；
- 为对象设置监听器，以响应用户操作。

1.6.1 引用组件

在activity中，可调用以下Activity方法引用已生成的组件：

```
public View findViewById(int id)
```

该方法以组件的资源ID作为参数，返回一个视图对象。

在QuizActivity.java文件中，使用按钮的资源ID获取视图对象，赋值给对应的成员变量，如代码清单1-8所示。注意，赋值前，必须先将返回的View类型转换为Button。

代码清单1-8 引用组件（QuizActivity.java）

```
public class QuizActivity extends AppCompatActivity {

    private Button mTrueButton;
    private Button mFalseButton;

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

        mTrueButton = (Button) findViewById(R.id.true_button);
        mFalseButton = (Button) findViewById(R.id.false_button);
    }
}
```

1.6.2 设置监听器

Android应用属于典型的事件驱动类型。不像命令行或脚本程序，事件驱动型应用启动后，即开始等待行为事件的发生，如用户点击某个按钮。（事件也可以由操作系统或其他应用触发，但用户触发的事件更直观，如点击按钮。）

应用等待某个特定事件的发生，也可以说应用正在“监听”特定事件。为响应某个事件而创建的对象叫作监听器（listener）。监听器会实现特定事件的监听器接口（listener interface）。

无需自己动手，Android SDK已经为各种事件内置了很多监听器接口。当前应用需要监听用户的按钮“点击”事件，因此监听器需实现View.OnClickListener接口。

首先处理TRUE按钮。在QuizActivity.java文件中，在onCreate(Bundle)方法的变量赋值语句后输入下列代码，如代码清单1-9所示。

代码清单1-9　为TRUE按钮设置监听器（QuizActivity.java）

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

        mTrueButton = (Button) findViewById(R.id.true_button);
        mTrueButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                // Does nothing yet, but soon!
            }
        });

        mFalseButton = (Button) findViewById(R.id.false_button);
    }
}
```

（如果遇到View cannot be resolved to a type的错误提示，请使用Option+Return（Alt+Enter）快捷键导入View类。）

在代码清单1-9中，我们设置了一个监听器。按钮mTrueButton被点击后，监听器会立即通知我们。传入setOnClickListener(OnClickListener)方法的参数是一个监听器。它是一个实现了OnClickListener接口的对象。

使用匿名内部类

这里，一个匿名内部类（anonymous inner class）实现了OnClickListener接口。语法看上去稍显复杂，不过有个助记小技巧：最外层一对括号内的全部代码就是传入setOnClickListener(OnClickListener)方法的参数。

```
mTrueButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        // Does nothing yet, but soon!
    }
});
```

本书所有的监听器都以匿名内部类来实现。这样做有两大好处。第一，使用匿名内部类，可以相对集中地实现监听器方法，一眼可见；第二，事件监听器一般只在一个地方使用，使用匿名内部类，就不用去创建繁琐的命名类了。

匿名内部类实现了OnClickListener接口，因此它也必须实现该接口唯一的onClick(View)方法。onClick(View)现在是个空方法。虽然必须实现onClick(View)方法，但具体如何实现取决于使用者，因此即使是个空方法，编译器也可以编译通过。

（如果匿名内部类、监听器、接口等概念已忘得差不多了，现在就该去复习一下，或找本参考手册备查。）

参照代码清单1-10为FALSE按钮设置类似的事件监听器。

代码清单1-10 为FALSE按钮设置监听器（QuizActivity.java）

```
        mTrueButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                // Does nothing yet, but soon!
            }
        });

        mFalseButton = (Button) findViewById(R.id.false_button);
        mFalseButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                // Does nothing yet, but soon!
            }
        });
    }
```

1.7 创建提示消息

接下来要实现的是，分别点击两个按钮，弹出我们称之为toast的提示消息。Android的toast是用来通知用户的简短弹出消息，用户无需输入什么，也不用做任何干预操作。这里，我们要用toast来反馈答案，如图1-14所示。

图1-14 toast消息反馈

首先回到strings.xml文件，如代码清单1-11所示，为toast添加消息显示用的字符串资源。

代码清单1-11　增加toast字符串（strings.xml）

```
<resources>
    <string name="app_name">GeoQuiz</string>
    <string name="question_text">Canberra is the capital of Australia.</string>
    <string name="true_button">True</string>
    <string name="false_button">False</string>
    <string name="correct_toast">Correct!</string>
    <string name="incorrect_toast">Incorrect!</string>
</resources>
```

调用Toast类的以下方法，可创建toast：

```
public static Toast makeText(Context context, int resId, int duration)
```

该方法的Context参数通常是Activity的一个实例（Activity本身就是Context的子类）。第二个参数是toast要显示字符串消息的资源ID。Toast类必须借助Context才能找到并使用字符串资源ID。第三个参数通常是两个Toast常量中的一个，用来指定toast消息的停留时间。

创建toast后，可调用Toast.show()方法在屏幕上显示toast消息。

在QuizActivity代码里，分别调用makeText(...)方法，如代码清单1-12所示。在添加makeText(...)时，可利用Android Studio的代码自动补全功能，让代码输入更轻松。

使用代码自动补全

代码自动补全功能可以节约大量开发时间，越早掌握受益越多。

参照代码清单1-12，依次输入代码。当输入到Toast类后的点号时，Android Studio会弹出一个窗口，窗口内显示了建议使用的Toast类的常量与方法。

要选择需要的建议方法，使用上下键。（如果不想使用代码自动补全功能，请不要按Tab键、Return/Enter键，或使用鼠标点击弹出窗口，只管继续输入代码直至完成。）

在建议列表里，选择makeText(Context context, int resID, int duration)方法，代码自动补全功能会自动添加完整的方法调用。

完成makeText方法的全部参数设置，完成后的代码如代码清单1-12所示。

代码清单1-12　创建提示消息（QuizActivity.java）

```
mTrueButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Toast.makeText(QuizActivity.this,
                       R.string.correct_toast,
                       Toast.LENGTH_SHORT).show();
            // Does nothing yet, but soon!
    }
});
mFalseButton = (Button) find ViewById(R.id.false_button);
mFalseButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
```

```
            Toast.makeText(QuizActivity.this,
                           R.string.incorrect_toast,
                           Toast.LENGTH_SHORT).show();
                // Does nothing yet, but soon!
        }
    });
```

在makeText(...)里，传入QuizActivity实例作为Context的参数值。注意此处应输入的参数是QuizActivity.this，不要想当然地直接输入this。因为匿名类的使用，这里的this指的是监听器View.OnClickListener。

使用代码自动补全功能，自己也就不用导入Toast类了，因为Android Studio会自动导入相关类。

好了，现在可以运行应用了。

1.8 使用模拟器运行应用

运行Android应用需使用硬件设备或虚拟设备（virtual device）。包含在开发工具中的Android设备模拟器可提供多种虚拟设备。

要创建Android虚拟设备（AVD），在Android Studio中，选择Tools → Android → AVD Manager菜单项。AVD管理器窗口弹出时，点击窗口左下角的+Create Virtual Device…按钮。

在随后弹出的对话框中，可以看到有很多配置虚拟设备的选项。对于首个虚拟设备，我们选择模拟运行Nexus 5X设备，如图1-15所示。点击Next继续。

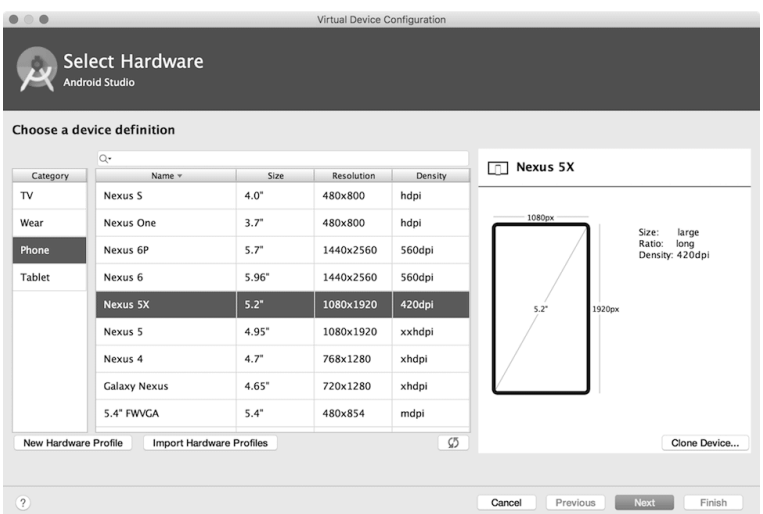

图1-15 创建新的AVD

如图1-16所示，接下来选择模拟器的系统镜像。选择x86 Nougat模拟器后点击Next按钮继续。

（在点击Next按钮之前，可能需要下载模拟器组件。按照提示操作即可。）

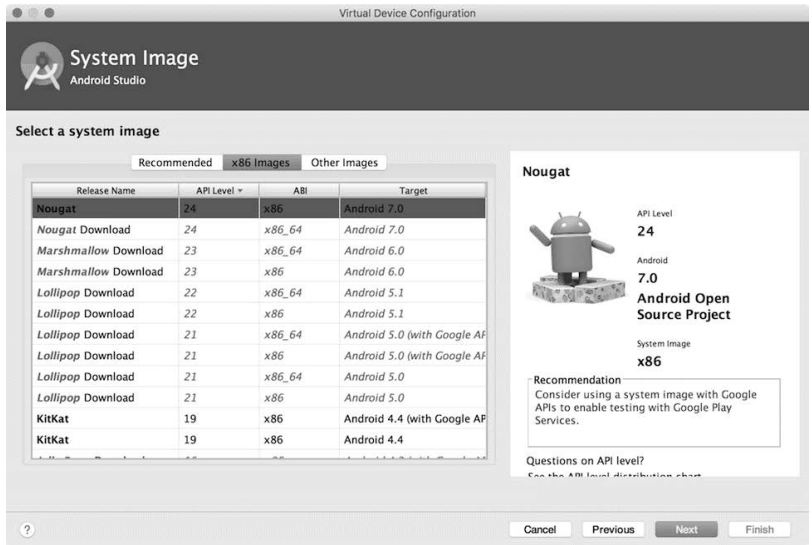

图1-16　选择系统镜像

最后，可以对模拟器的各项参数做最终修改并确认，如图1-17所示。当然，如果有需要，也可以事后再编辑修改模拟器的各项参数。现在，为模拟器取个便于识别的名称，然后点击Finish按钮完成虚拟设备的创建。

图1-17　模拟器参数调整

用重建了），而是调用了onStart()和onResume()方法。用户按了主屏幕键后，QuizActivity最后进入停止状态，再次调出应用时，QuizActivity只需要重新启动（进入暂停状态，用户可见），然后继续运行（进入运行状态，活动在前台）。

activity有时也会一直处于暂停状态（用户完全或部分可见，但不在前台）。可能出现部分可见暂停状态的场景：在一个activity之上启动带透明背景视图或小于屏幕尺寸视图的新activity时。可能出现完全可见暂停状态的场景：应用多窗口模式下（Android 6.0及更高系统版本才支持），当前activity在一个窗口完全可见，而用户在不包含当前activity的另一个窗口操作时。

在本书的后续学习过程中，为完成各种现实任务，我们还会覆盖一些其他生命周期方法，进一步学习更多生命周期方法的用法。

3.2　设备旋转与 activity 生命周期

现在，可以处理第2章结束时发现的应用缺陷了。启动GeoQuiz应用，单击NEXT按钮显示第二道地理知识问题，然后旋转设备。（模拟器的旋转，使用Command+右方向键/Ctrl+右方向键，或点击工具栏上的旋转按钮。）

设备旋转后，GeoQuiz应用又回到了第一道问题。查看LogCat日志看看发生了什么，如图3-9所示。

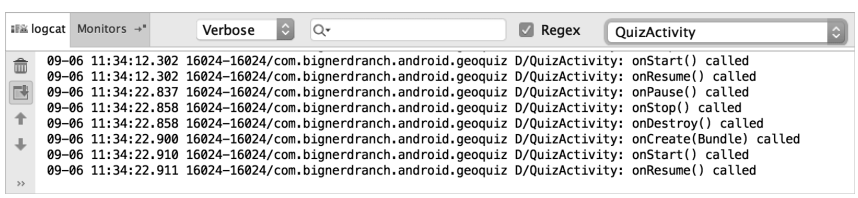

图3-9　QuizActivity已死，QuizActivity万岁

设备旋转时，系统会销毁当前QuizActivity实例，然后创建一个新的QuizActivity实例。再次旋转设备，又一次见证这个销毁与再创建的过程。

这就是问题所在。每次旋转设备，当前QuizActivity实例会完全销毁，实例中的mCurrentIndex当前值会从内存里被抹掉。旋转后，Android重新创建了QuizActivity新实例，mCurrentIndex在onCreate(Bundle)方法中被初始化为0。一切重头再来，用户又看到第一道题。

稍后会修正这个缺陷。不要停留在问题表面，接下来，由表及里，深入分析为什么有此问题。

设备配置与备选资源

旋转设备会改变设备配置（device configuration）。设备配置实际是一系列特征组合，用来描述设备当前状态。这些特征有：屏幕方向、屏幕像素密度、屏幕尺寸、键盘类型、底座模式以及语言等。

通常，为匹配不同的设备配置，应用会提供不同的备选资源。为适应不同分辨率的屏幕，向

项目添加多套箭头图标就是这样一个使用案例。

设备的屏幕像素密度是个固定的设备配置，无法在运行时发生改变。然而，屏幕方向等特征可以在应用运行时改变。

在运行时配置变更（runtime configuration change）发生时，可能会有更合适的资源来匹配新的设备配置。于是，Android销毁当前activity，为新配置寻找最佳资源，然后创建新实例使用这些资源。眼见为实，下面为设备配置变更新建备选资源，只要设备旋转至水平方位，Android就会自动发现并使用它。

创建水平模式布局

在项目工具窗口中，右键单击res目录后选择New → Android resource directory菜单项。创建资源目录界面列出了资源类型及其对应的资源特征，如图3-10所示。从资源类型（Resource type）列表中选择layout，保持Source set的main选项不变。

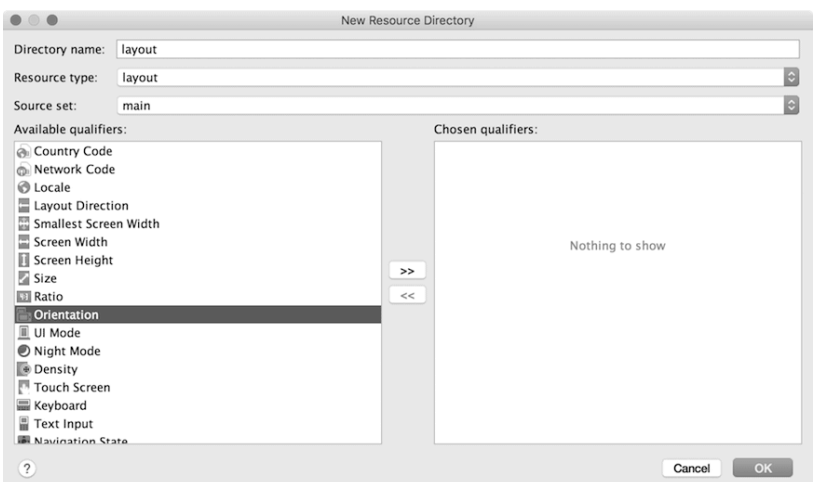

图3-10 创建新的资源目录

接下来选中待选资源特征列表中的Orientation，然后单击>>按钮将其移动至已选资源特征（Chosen qualifiers）区域。

最后，确认选中Screen orientation下拉列表中的Landscape选项，并确保目录名（Directory name）显示为layout-land，如图3-11所示。这个窗口看起来有模有样，但实际用途仅限于设置目录名。点击OK按钮让Android Studio创建res/layout-land。

这里的-land后缀名是配置修饰符的另一个使用例子。Android依靠res子目录的配置修饰符定位最佳资源以匹配当前设备配置。访问Android开发网页http://developer.android.com/guide/topics/resources/providing-resources.html，可查看Android的配置修饰符列表及其代表的设备配置信息。

设备处于水平方向时，Android会找到并使用res/layout-land目录下的布局资源。其他情况下，它会默认使用res/layout目录下的布局资源。然而，目前在res/layout-land目录下并没有布局资源。让我们解决这个问题。

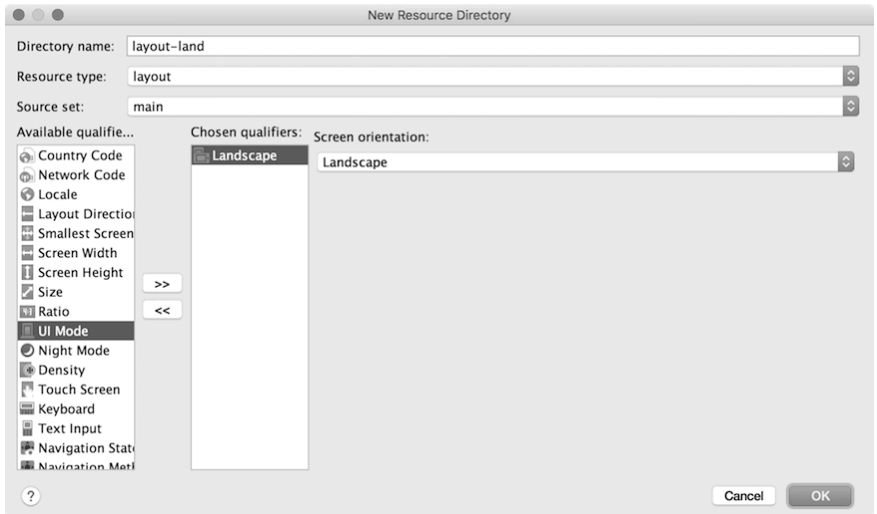

图3-11　创建res/layout-land

从res/layout目录复制activity_quiz.xml文件至res/layout-land目录。(如果在Android工具窗口看不到res/layout-land目录，请从Android视图切换到Project视图。完成后，记得切回。如果喜欢，你也可以直接使用文件管理器或终端应用复制粘贴文件。)

现在我们有了一个水平模式布局以及一个默认布局 (竖直模式)。注意，两个布局文件的文件名必须相同，这样它们才能以同一个资源ID被引用。

为了与默认的布局文件相区别，还需修改水平模式布局文件。请参照图3-12进行相应修改。

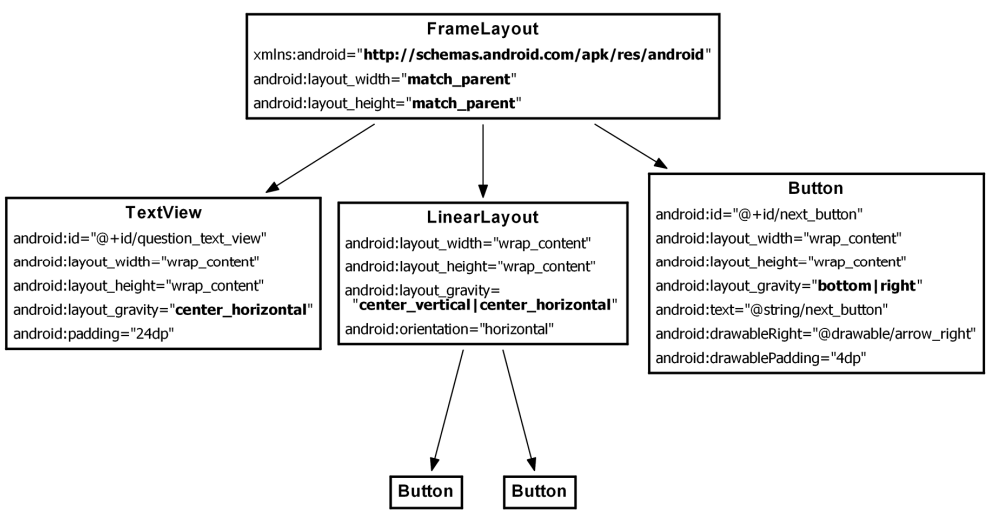

图3-12　备选的水平模式布局

如图3-12所示，FrameLayout替换了最上层的LinearLayout。FrameLayout是最简单的ViewGroup组件，它一概不管如何安排其子视图的位置。FrameLayout子视图的位置排列取决于它们各自的android:layout_gravity属性。

因而，TextView、LinearLayout和Button都需要一个android:layout_gravity属性。这里，LinearLayout里的Button子元素保持不变。

参照图3-12，打开layout-land/activity_quiz.xml文件修改。完成后可同代码清单3-4做对比检查。

代码清单3-4 水平模式布局修改（layout-land/activity_quiz.xml）

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

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

    <TextView
        android:id="@+id/question_text_view"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center_horizontal"
        android:padding="24dp" />

    <LinearLayout
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center_vertical|center_horizontal"
        android:orientation="horizontal" >
      ...
    </LinearLayout>

    <Button
        android:id="@+id/next_button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="bottom|right"
        android:text="@string/next_button"
        android:drawableRight="@drawable/arrow_right"
        android:drawablePadding="4dp"
        />

</LinearLayout>
</FrameLayout>
```

再次运行GeoQuiz应用。旋转设备至水平方位，查看新的布局界面，如图3-13所示。当然，这不仅是一个新的布局界面，也是一个新的QuizActivity。