第 3 章

Activity的生命周期

每个Activity实例都有其生命周期。在其生命周期内，activity在运行、暂停和停止三种可能的状态间进行转换。每次状态发生转换时，都有一个Activity方法将状态改变的消息通知给activity。图3-1显示了activity的生命周期、状态以及状态切换时系统调用的方法。

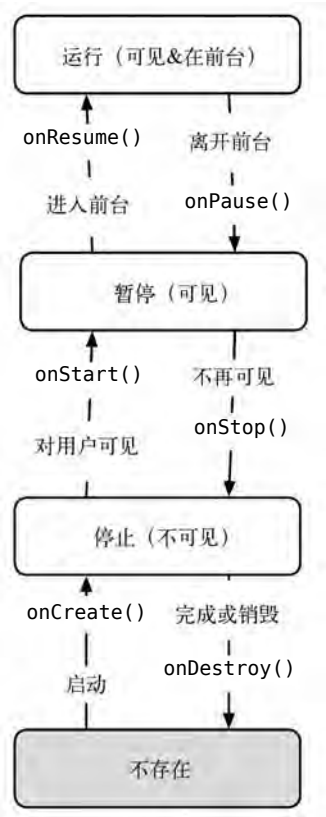

图3-1　Activity的状态图解

利用图3-1所示的方法，Activity的子类可以在activity的生命周期状态发生关键性转换时完

成某些工作。

我们已经熟悉了这些方法中的onCreate(Bundle)方法。在创建activity实例后，但在此实例出现在屏幕上以前，Android操作系统会调用该方法。

通常，activity通过覆盖onCreate(...)方法来准备以下用户界面的相关工作：

- 实例化组件并将组件放置在屏幕上（调用方法setContentView(int)）；
- 引用已实例化的组件；
- 为组件设置监听器以处理用户交互；
- 访问外部模型数据。

无需自己调用onCreate(...)方法或任何其他Activity生命周期方法，理解这一点很重要。我们只需在activity子类里覆盖这些方法即可，Android会适时去调用它们。

3.1　日志跟踪理解 Activity 生命周期

本节将通过覆盖activity生命周期方法的方式，来探索QuizActivity的生命周期。在每一个覆盖方法的具体实现里，输出的日志信息都表明了当前方法已被调用。

3.1.1　输出日志信息

Android内部的android.util.log类能够发送日志信息到系统级别的共享日志中心。Log类有好几个日志信息记录方法。本书使用最多的是以下方法：

```
public static int d(String tag, String msg)
```

d代表着“debug”的意思，用来表示日志信息的级别。（本章最后一节将会更为详细地为大家讲解有关Log级别的内容。）第一个参数表示日志信息的来源，第二个参数表示日志的具体内容。

该方法的第一个参数通常以类名为值的TAG常量传入。这样，很容易看出日志信息的来源。

在QuizActivity.java中，为QuizActivity类新增一个TAG常量，如代码清单3-1所示。

代码清单3-1　新增一个TAG常量（QuizActivity.java）

```
public class QuizActivity extends Activity {

    private static final String TAG = "QuizActivity";

    ...

}
```

然后，在onCreate(...)方法里调用 Log.d(...)方法记录日志信息，如代码清单3-2所示。

代码清单3-2　为onCreate(...)方法添加日志输出代码（QuizActivity.java）

```
public class QuizActivity extends Activity {

    ...

    @Override
```

```
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        Log.d(TAG, "onCreate(Bundle) called");
        setContentView(R.layout.activity_quiz);

        ...
```

参照代码清单3-2输入相应代码，Eclipse可能会提示无法识别Log类的错误。这时，记得使用Mac系统的Command+Shift+O或者Windows系统的Ctrl+Shift+O组合键进行类包组织导入。在Eclipse询问引入哪个类时，选择android.util.Log类。

接下来，在QuizActivity类中，继续覆盖其他五个生命周期方法，如代码清单3-3所示。

代码清单3-3 覆盖更多生命周期方法（QuizActivity.java）

```
    } // End of onCreate(Bundle)

    @Override
    public void onStart() {
        super.onStart();
        Log.d(TAG, "onStart() called");
    }

    @Override
    public void onPause() {
        super.onPause();
        Log.d(TAG, "onPause() called");
    }

    @Override
    public void onResume() {
        super.onResume();
        Log.d(TAG, "onResume() called");
    }

    @Override
    public void onStop() {
        super.onStop();
        Log.d(TAG, "onStop() called");
    }

    @Override
    public void onDestroy() {
        super.onDestroy();
        Log.d(TAG, "onDestroy() called");
    }

}
```

请注意，我们先是调用了超类的实现方法，然后再调用具体日志的记录方法。调用这些超类方法必不可少。在onCreate(...)方法里，必须先调用超类的实现方法，然后再调用其他方法，这一点很关键。而在其他方法中，是否首先调用超类方法就不那么重要了。

为何要使用@Override注解呢？可能一直以来你都对此感到非常困惑。使用@Override注解，即要求编译器保证当前类具有准备覆盖的方法。例如，对于如下代码中名称拼写错误的方法，编译器将发出警告：

```
public class QuizActivity extends Activity {

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

    ...
```

由于Activity类中不存在onCreat(Bundle)方法，因此编译器发出了警告。这样就可以改正拼写错误，而不是碰巧实现了一个名为QuizActivity.onCreat(Bundle)的方法。

3.1.2　使用 LogCat

应用运行时，可以使用LogCat工具来查看日志。Logcat是Android SDK工具中的日志查看器。

要想打开LogCat，可选择Window → Show View → Other...菜单项。在随后弹出的对话框中，展开Android文件夹找到并选择LogCat，然后单击OK按钮，如图3-2所示。

图3-2　寻找 LogCat

LogCat窗口出现在屏幕的右半边。恼人的是，这使得编辑区可视区域变小了很多。要是能把它放置在工作区窗口的底部就好了。

单击并按住LogCat窗口的标签空白处，可把它拖曳到工作区窗口的右下角工具栏上，如图3-3所示。

此时，LogCat窗口将会关闭，同时它的图标会出现在底部的工具栏上。点击图标，在工作区窗口底部重新打开它。

现在，Eclipse工作区看起来应该如图3-4一样。拖曳LogCat窗口的边框可以调整窗口的区域大小。该操作对Eclipse工作区的其他窗口同样适用。图3-4为LogCat被拖放到工具栏后的Elipse工作区。

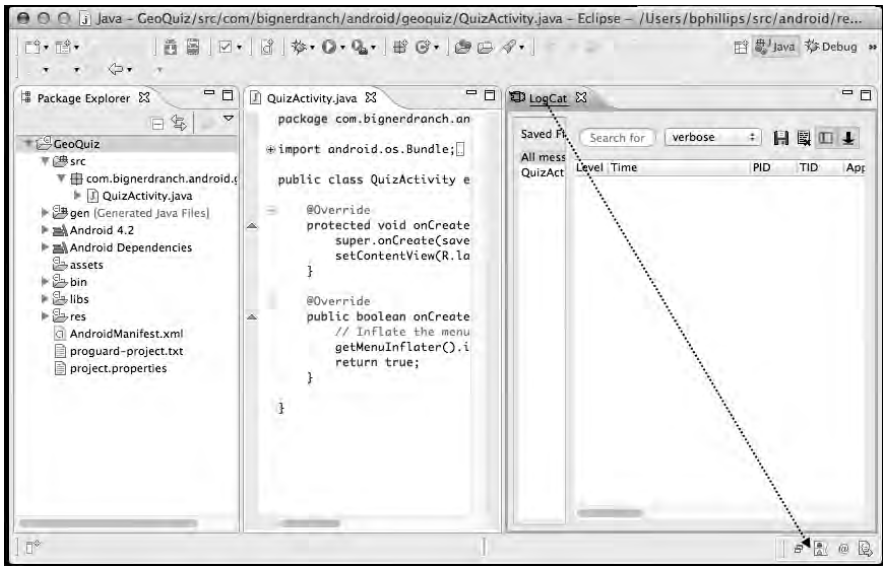

图3-3 拖曳LogCat标签到右下角工具栏上

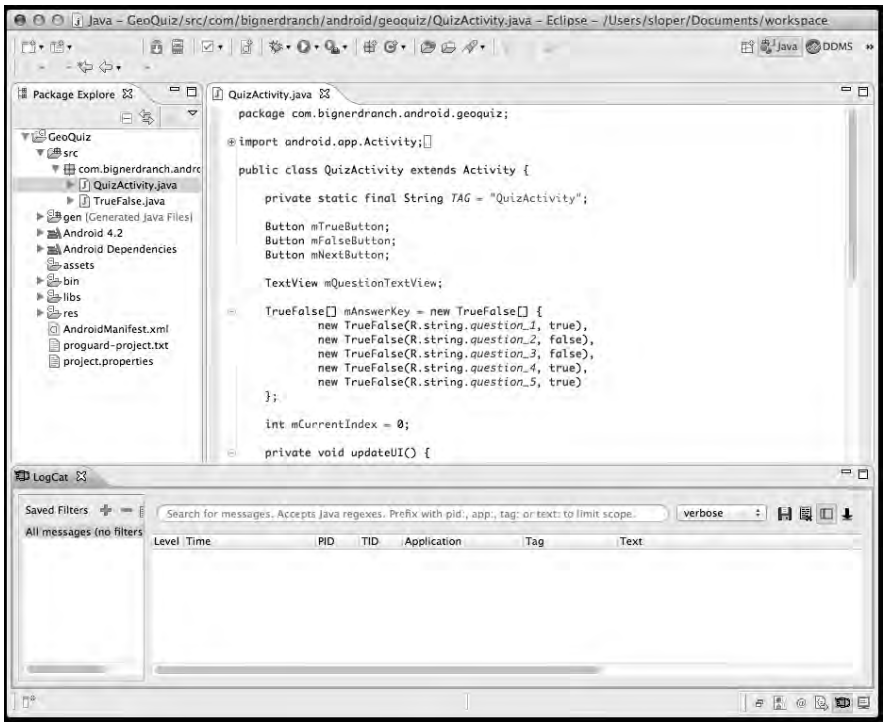

图3-4 LogCat被拖放到工具栏后的Eclipse工作区

运行GeoQuiz应用。急速翻滚的各类信息立即出现在LogCat窗口中。其中大部分信息都来自于系统的输出。滚动到该日志窗口的底部查找想看的日志信息。在LogCat的Tag列，可看到为`QuizActivity`类创建的TAG常量。

（如无法看到任何日志信息，很可能是因为LogCat正在监控其他设备。选择Window → Show View → Other...菜单项，打开Devices视图，选中要监控的设备后再切换回LogCat。）

为方便日志信息的查找，可使用TAG常量过滤日志输出。单击LogCat左边窗口上方的绿色+按钮，创建一个消息过滤器。Filter Name输入`QuizActivity`，by Log Tag同样输入`QuizActivity`，如图3-5所示。

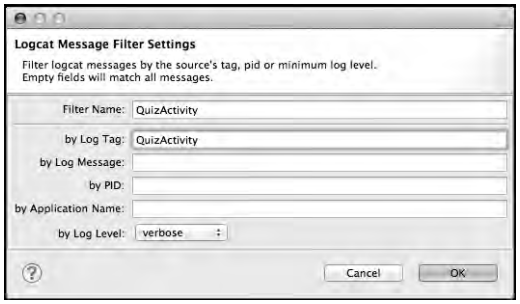

图3-5　在LogCat中创建过滤器

单击OK按钮，在新出现的标签页窗口中，仅显示了Tag为`QuizActivity`的日志信息，如图3-6所示。日志里可以看到，GeoQuiz应用启动并完成`QuizActivity`初始实例创建后，有三个生命周期方法被调用了。

图3-6　应用启动后，被调用的三个生命周期方法

（如看不到过滤后的信息列表，请选择LogCat左边窗口的`QuizActivity`过滤项。）

现在我们来做个有趣的实验。在设备上单击后退键，再查看LogCat。可以看到，日志显示`QuizActivity`的`onPause()`、`onStop()`和`onDestroy()`方法被调用了，如图3-7所示。

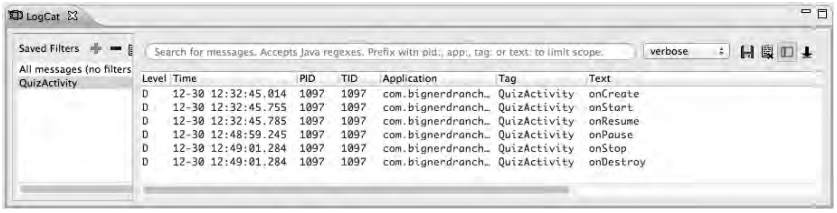

图3-7　单击后退键销毁activity

单击设备的后退键，相当于通知Android系统“我已完成acitivity的使用，现在不需要它了。”接到指令后，系统立即销毁了acitivity。这实际是Android系统节约使用设备有限资源的一种方式。

重新运行GeoQuiz应用。这次，选择单击主屏幕键，然后查看LogCat。日志显示系统调用了QuizActivity的`onPause()`和`onStop()`方法，但并没有调用`onDestroy()`方法，如图3-8所示。

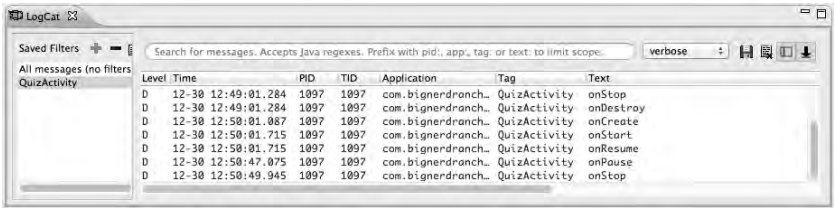

图3-8 单击主屏幕键停止activity

要在设备上调出任务管理器，如果是比较新的设备，可单击主屏幕键旁的最近应用键，调出任务管理器，如图3-9所示。如果设备没有最近应用键，则长按主屏幕键调出任务管理器。

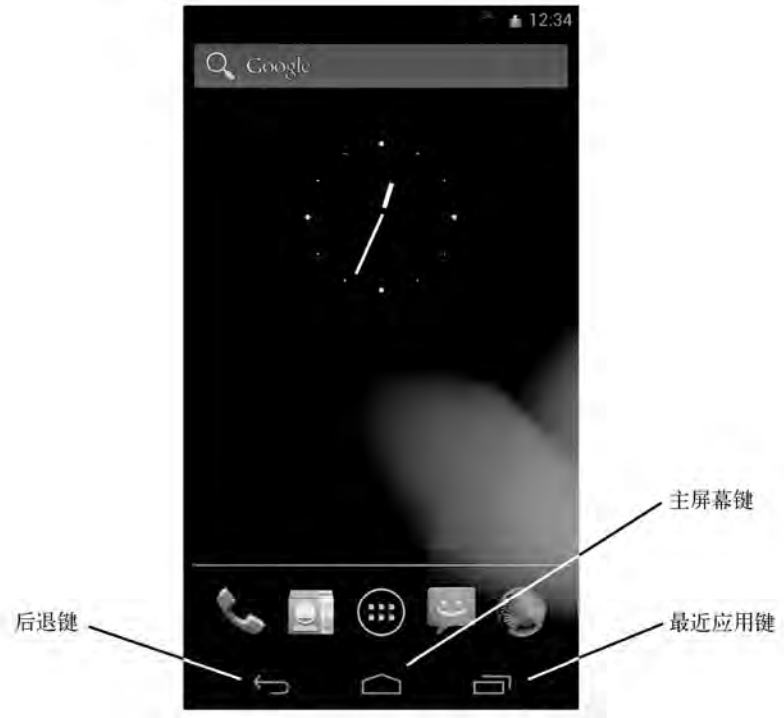

图3-9 主屏幕键，后退键以及最近应用键

在任务管理器中，单击GeoQuiz应用，然后查看LogCat。日志显示，activity无需新建即可启动并重新开始运行。

单击主屏幕键，相当于通知Android“我去别处看看，稍后可能回来。”此时，为快速响应随时返回应用，Android只是暂停当前activity而并不销毁它。

需要注意的是，停止的activity能够存在多久，谁也无法保证。如果系统需要回收内存，它将首先销毁那些停止的activity。

最后，想象一下存在一个会部分遮住当前activity界面的小弹出窗口。它出现时，被遮住的activity会被系统暂停，用户也无法同它交互。它关闭时，被遮住的activity将会重新开始运行。

在本书的后续学习过程中，为完成各种实际的任务，需覆盖不同的生命周期方法。通过这样不断地实践，我们将学习到更多使用生命周期方法的知识。

3.2　设备旋转与 Activity 生命周期

现在，我们来处理第2章结束时发现的应用缺陷。运行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Control+F12/Ctrl+F12组合键。）

设备旋转后，GeoQuiz应用又重新显示了第一道问题。查看LogCat日志查找问题原因，如图3-10所示。

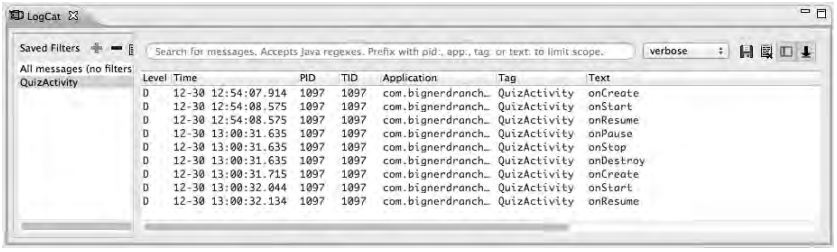

图3-10　QuizActivity已死，QuizActivity万岁

设备旋转时，当前看到的QuizActivity实例会被系统销毁，然后创建一个新的QuizActivity实例。再次旋转设备，查看该销毁与再创建的过程。

这就是问题产生的原因。每次创建新的QuizActivity实例时，mCurrentIndex会被初始化为0，因此用户又回到了第一道问题上。稍后我们会修正这个缺陷。现在先来深入分析下该问题产生的原因。

设备配置与备选资源

旋转设备会改变设备配置（device configuration）。设备配置是用来描述设备当前状态的一系列特征。这些特征包括：屏幕的方向、屏幕的密度、屏幕的尺寸、键盘类型、底座模式以及语言，等等。

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，向项目里添加多套箭头图标就是这样一个使用案例。

设备的屏幕密度是一个固定的设备配置，无法在运行时发生改变。然而，有些特征，如屏幕方向，可以在应用运行时进行改变。

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。眼见为实，下面新建一个备选资源，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

首先，最小化LogCat窗口。（如果不小心关掉了Logcat，可选择Window→Show View...菜单项重新打开它。）

然后，在包浏览器中，右键单击res目录创建一个新文件夹并命名为layout-land，如图3-11所示。

图3-11 创建新文件夹

将activity_quiz.xml文件从res/layout/目录复制至res/layout-land/目录。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，两个布局文件必须具有相同的文件名，这样它们才能以同一个资源ID被引用。

这里的-land后缀名是配置修饰符的另一个使用例子。res子目录的配置修饰符表明了Android是如何通过它来定位最佳资源以匹配当前设备配置的。访问Android开发网页http://developer.android.com/guide/topics/resources/providing-resources.html，可查看Android的配置修饰符列表以及配置修饰符代表的设备配置信息。第15章将有更多机会练习使用这些配置修饰符。

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，会默认使用res/layout目录下的布局资源。

为与默认的布局文件相区别，我们需要对水平模式布局文件做出一些修改。图3-12显示了将要对默认资源文件做出的修改。

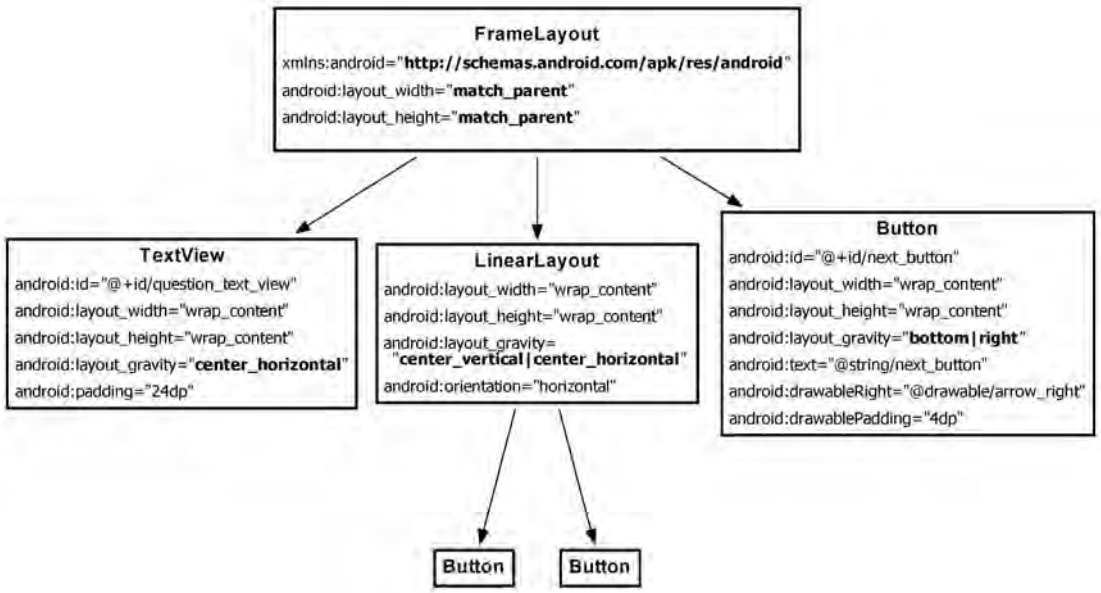

图3-12　备选的水平模式布局

用FrameLayout替换LinearLayout。FrameLayout是一种最简单的ViewGroup组件，它不以特定方式安排其子视图的位置。FrameLayout子视图的位置排列都是由它们各自的android:layout_gravity属性决定的。

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文件进行相应的修改。然后使用代码清单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QeoQuiz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，这不仅仅是一个新的布局界面，也是一个新的QuizActivity。

图3-13 处于水平方位的QuizActivity

设备旋转回竖直方位，可看到默认的布局界面以及另一个新的QuizActivity。

Android可自动完成调用最佳匹配资源的工作，但前提是它必须通过新建一个activity来实现。QuizActivity要显示一个新布局，方法setContentView(R.layout.activity_quiz)必须再次被调用。而调用setContentView(R.layout.activity_quiz)方法又必须先调用QuizActivity.onCreate(...)方法。因此，设备一经旋转，Android需要销毁当前的QuizActivity，然后再新

建一个QuizActivity来完成QuizActivity.onCreate(...)方法的调用，从而实现使用最佳资源匹配新的设备配置。

请记住，只要在应用运行中设备配置发生了改变，Android就会销毁当前activity，然后再新建一个activity。另外，在应用运行中，虽然也会发生可用键盘或语言的改变，但设备屏幕方向的改变是最为常见的情况。

3.3　设备旋转前保存数据

适时使用备选资源虽然是Android提供的较完美的解决方案。但是，设备旋转导致的activity销毁与新建也会带来麻烦。比如，设备旋转后，GeoQuiz应用回到第一道题目的缺陷。

要修正这个缺陷，旋转后新创建的QuizActivity需要知道mCurrentIndex变量的原有值。因此，在设备运行中发生配置变更时，如设备旋转，需采用某种方式保存以前的数据。覆盖以下Activity方法就是一种实现方式：

```
protected void onSaveInstanceState(Bundle outState)
```

该方法通常在onPause()、 onStop()以及onDestroy()方法之前由系统调用。

方法onSaveInstanceState(...)默认的实现要求所有activity的视图将自身状态数据保存在Bundle对象中。Bundle是存储字符串键与限定类型值之间映射关系（键–值对）的一种结构。

之前已使用过Bundle，如下列代码所示，它作为参数传入onCreate(Bundle)方法：

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    ...
```

覆盖onCreate(...)方法时，我们实际是在调用activity超类的onCreate(...)方法，并传入收到的bundle。在超类代码实现里，通过取出保存的视图状态数据，activity的视图层级结构得以重新创建。

覆盖onSaveInstanceState(Bundle)方法

可通过覆盖onSaveInstanceState(...)方法，将一些数据保存在Bundle中，然后在onCreate(...)方法中取回这些数据。设备旋转时，将采用这种方式保存mCurrentIndex变量值。

首先，打开QuizActivity.java文件，新增一个常量作为将要存储在bundle中的键-值对的键，如代码清单3-5所示。

代码清单3-5　新增键–值对的键（QuizActivity.java）

```
public class QuizActivity extends Activity {

    private static final String TAG = "QuizActivity";
    private static final String KEY_INDEX = "index";

    Button mTrueButton;
    ...
```

然后，覆盖onSaveInstanceState(...)方法，以刚才新增的常量值作为键，将mCurrentIndex变量值保存到Bundle中，如代码清单3-6所示。

代码清单3-6 覆盖onSaveInstanceState(...)方法（QuizActivity.java）

```
        mNextButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                mCurrentIndex = (mCurrentIndex + 1) % mQuestionBank.length;
                updateQuestion();
            }
        });

        updateQuestion();
    }

    @Override
    public void onSaveInstanceState(Bundle savedInstanceState) {
        super.onSaveInstanceState(savedInstanceState);
        Log.i(TAG, "onSaveInstanceState");
        savedInstanceState.putInt(KEY_INDEX, mCurrentIndex);
    }
```

最后，在onCreate(...)方法中查看是否获取了该数值。如确认获取成功，则将它赋值给变量mCurrentIndex，如代码清单3-7所示。

代码清单3-7 在onCreate(...)方法中检查存储的bundle信息（QuizActivity.java）

```
        ...

        if (savedInstanceState != null) {
            mCurrentIndex = savedInstanceState.getInt(KEY_INDEX, 0);
        }

        updateQuestion();
    }
```

运行GeoQuiz应用。单击下一步按钮。现在，无论设备自动或手动旋转多少次，新创建的QuizActivity都将会记住当前正在回答的题目。

注意，我们在Bundle中存储和恢复的数据类型只能是基本数据类型（primitive type）以及可以实现Serializable接口的对象。创建自己的定制类时，如需在onSaveInstanceState(...)方法中保存类对象，记得实现Serializable接口。

测试onSaveInstanceState(...)的实现是个好习惯，尤其在需要存储和恢复对象时。设备旋转很容易测试，但测试低内存状态就困难多了。本章末尾会深入学习这部分内容，继而学习如何模拟Android为回收内存而销毁activity的场景。

3.4 再探 Activity 生命周期

覆盖onSaveInstanceState(...)方法并不仅仅用于处理设备旋转相关的问题。用户离开当前acitivity管理的用户界面，或Android需要回收内存时，activity也会被销毁。

不过Android从不会为了回收内存，而去销毁正在运行的activity。activity只有在暂停或停止状态下才可能会被销毁。此时，会调用onSaveInstanceState(...)方法。

调用onSaveInstanceState(...)方法时，用户数据随即被保存在Bundle对象中。然后操作系统将Bundle对象放入activity记录中。

为便于理解activity记录，我们增加一个暂存状态（stashed state）到activity生命周期，如图3-14。

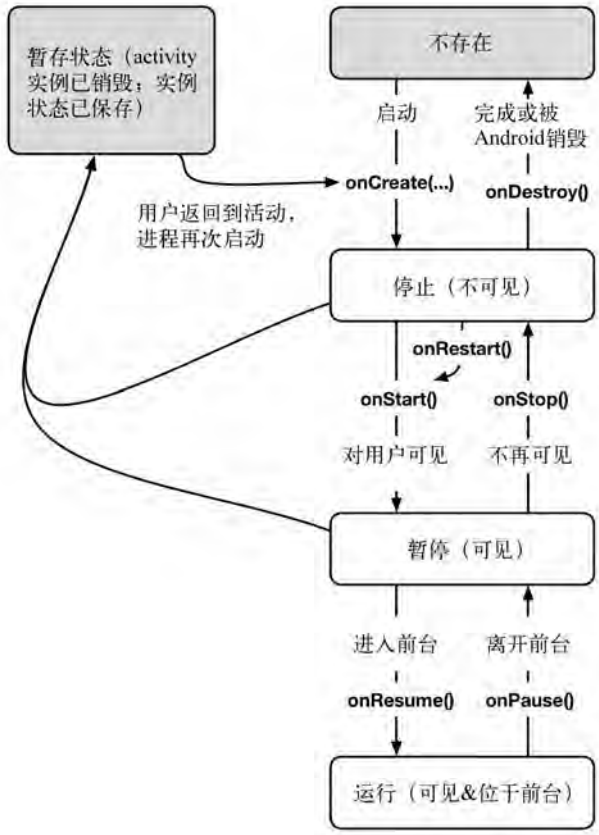

图3-14　完整的activity生命周期

activity暂存后，Activity对象不再存在，但操作系统会将activity记录对象保存起来。这样，在需要恢复activity时，操作系统可以使用暂存的activity记录重新激活activity。

注意，activity进入暂存状态并不一定需要调用onDestroy()方法。不过，onPause()和onSaveInstanceState(...)通常是我们需要调用的两个方法。常见的做法是，覆盖onSaveInstanceState(...)方法，将数据暂存到Bundle对象中，覆盖onPause()方法处理其他需要处理的事情。

有时，Android不仅会销毁activity，还会彻底停止当前应用的进程。不过，只有在用户离开当前应用时才会发生这种情况。即使这种情况真的发生了，暂存的activity记录依然被系统保留着，

以便于用户返回应用时activity的快速恢复。

那么暂存的activity记录到底可以保留多久？前面说过，用户按了后退键后，系统会彻底销毁当前的activity。此时，暂存的activity记录同时被清除。此外，系统重启或长时间不使用activity时，暂存的activity记录通常也会被清除。

3.5 深入学习：测试 onSaveInstanceState(Bundle)方法

覆盖onSaveInstanceState(Bundle)方法时，应测试activity状态是否如预期正确保存和恢复。使用模拟器很容易做到这些。

启动虚拟设备。在设备应用列表中找到Settings应用，如图3-15所示。大部分模拟器包含的系统镜像应该都包含该应用。

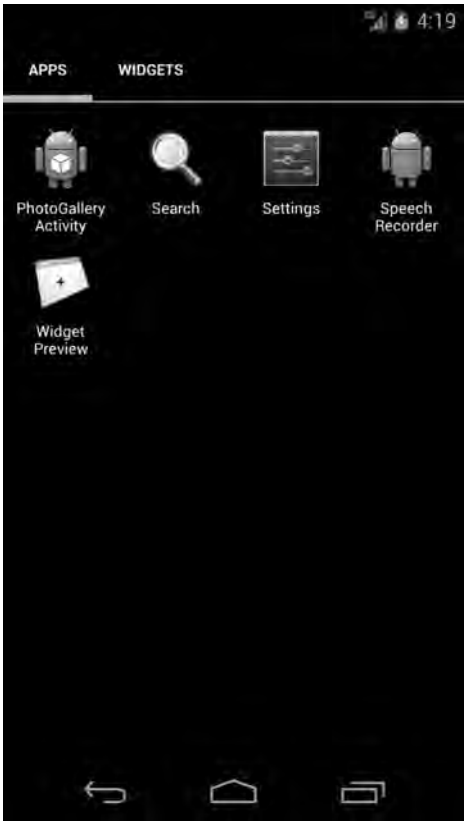

图3-15 找到Settings应用

启动Settings应用，点击Development options选项，找到并启用Don’t keep activities选项，如图3-16所示。

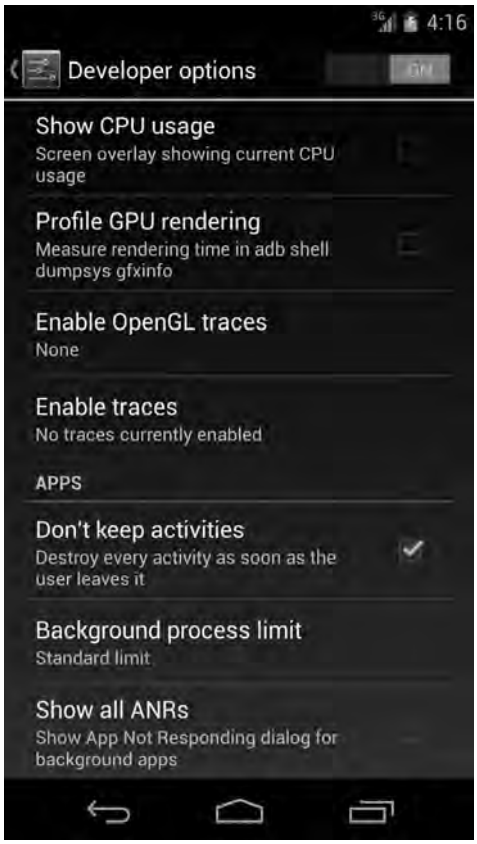

图3-16　启用Don’t keep activities选项

现在运行应用，单击主屏幕键（如前所述，点击主屏幕键会暂停并停止当前activity）。随后就像Android 操作系统为回收内存一样，停止的activity被系统销毁了。可通过重新运行应用，验证activity状态是否如期得到保存。

和单击主屏幕键不一样的是，单击后退键后，无论是否启用Don’t keep activities选项，系统总是会销毁当前activity。单击后退键相当于通知系统“用户不再需要使用当前的activity”。

如需在硬件设备上进行同样的测试，必须安装额外的开发工具。请访问http://developer.android.com/tools/debugging/debugging-devtools.html了解详情。

3.6　深入学习：日志记录的级别与方法

使用`android.util.Log`类记录日志信息，不仅可以控制日志信息的内容，还可以控制用来划分信息重要程度的日志级别。Android支持如图3-17所示的五种日志级别。每一个级别对应着一个`Log`类方法。调用对应的`Log`类方法与日志的输出和记录一样容易，如图3-17所示。

Log Level	Method	说 明
ERROR	Log.e(...)	错误
WARNING	Log.w(...)	警告
INFO	Log.i(...)	信息型消息
DEBUG	Log.d(...)	调试输出：可能被过滤掉
VERBOSE	Log.v(...)	只用于开发

图3-17 日志级别与方法

需要说明的是，所有的日志记录方法都有两种参数签名：string类型的tag参数和msg参数；除tag和msg参数外再加上Throwable实例参数。附加的Throwable实例参数为应用抛出异常时记录异常信息提供了方便。代码清单3-8展示了两种方法不同参数签名的使用实例。对于输出的日志信息，可使用常用的Java字符串连接操作拼接出需要的信息。或者使用String.format对输出日志信息进行格式化操作，以满足个性化的使用要求。

代码清单3-8 Android的各种日志记录方式

```
// Log a message at "debug" log level
Log.d(TAG, "Current question index: " + mCurrentIndex);

TrueFalse question;
try {
    question = mQuestionBank[mCurrentIndex];
} catch (ArrayIndexOutOfBoundsException ex) {
    // Log a message at "error" log level, along with an exception stack trace
    Log.e(TAG, "Index was out of bounds", ex);
}
```