广东工业大学

华立学院

# 本科文献翻译

外文参考文献译文及原文

学 部 信息与工程

专 业 计算机科学与技术

年 级 2013级

班级名称 13计算机1班

学 号 511321010110

学生姓名 张松周

指导教师 程东胜

2017年 4月 18日

**目 录**

[（1）Android SDK: Activities & Lifecycle 1](#_Toc15980)

[Introduction: 1](#_Toc9566)

[（1）Activity 与生命周期 4](#_Toc21363)

[介绍 4](#_Toc12521)

[（2）Android SDK: App Structure 7](#_Toc28699)

[Introduction 7](#_Toc16312)

[（2）Android 应用程序结构 11](#_Toc11640)

[介绍 11](#_Toc485)

（1）Android SDK: Activities & Lifecycle

Introduction:

The Activity life cycle doesn't just come into play when the user runs your app, it also affects what happens when the user switches away from and back to your app. When you develop an app, you need to remember that the user will typically navigate between it and other apps during execution. An app can either be in the foreground or the background depending on what the user is doing. Your apps need to be able to cope with this, saving and restoring data when necessary. Keep in mind that the process is slightly different for some apps, like widgets.

1:Callback Methods

Step 1:

To control what happens when your Activity is in different states, for example when the user navigates away from or back to it, you can implement various methods. These are the Activity life cycle callback methods. Android calls these methods when your Activity enters each state, so you can take steps to ensure the app continues to function, does not lose data, and does not use unnecessary resources when the user is not interacting with it. Each callback method puts your app in one of a range of possible states.

If you've ever programmed Java applications before, you know that Android apps are launched differently. Rather than using a main method, when your app is launched the onCreate method of the main Activity class executes. Remember that we specified the class in the Manifest as the main launcher Activity. The onCreate method is the first of the Activity callbacks, which executes when the user launches your app. The onCreate method puts your app in the Created state.

The diagram in the Developer Guide provides an intuitive representation of the life cycle, callback methods, and states. The Resumed state, which represents your app running as the user interacts with it, follows the onResume method. Various callback methods take your app towards and away from the Resumed state, starting and stopping it.

For most apps you only need to implement some of the callback methods, at the very least onCreate. However, understanding all of the callbacks and states will help you to grasp what happens on the Android system when your app runs and stops running. In general, you need to implement whichever callbacks that let the user resume your app from where they left off, and if they navigate away and back, save any data you need to persist and prevent your app from using resources unnecessarily.

Step 2

There are five states your app can be in: Created, Started, Resumed, Paused, and Stopped. Seven callback methods transition into and out of these states: onCreate, onStart, onRestart, onResume, onPause, onStop, and onDestroy. These methods take your app between the possible states, in some cases very quickly. In general you can think of your app as being either resumed, paused, or stopped at any time, since the other states are transitory.

When your app is running and the user is interacting with it, it is in the Resumed state. Your app goes into Paused state when another Activity is in the foreground but only partially hides it; the user can't interact with your app in this state. When your app is completely in the background and the user can't see or interact with it at all, it is in the Stopped state. Here the Activity can retain data in this state but can't execute.

2. Moving Towards Resumed

As we know, the main Activity starts when the app is launched, with the onCreate method executing, letting you prepare the Activity UI and any data items you need throughout the class. Most apps you build will have more than one Activity in them, with the other Activities launched on user interaction with the app. You can launch one Activity from another using the Intent class as in the following example code:

Intent aboutIntent = new Intent(this, About.class);

startActivity(aboutIntent);

This represents another Activity class named "About" in your app package. You can create a new Activity in Eclipse by selecting your source package and choosing "File", "New", "Class", and selecting the Android Activity class as superclass. Remember that each Activity must be listed in your Manifest. You can also pass data between Activities using the Intent class.

When any Activity runs, the onCreate method executes, so you can treat other Activity classes in your apps similarly to the main Activity except for the way you list them in the Manifest. You can create a layout file for each Activity and set it using the same technique you used for the main Activity.

After the onCreate method of an Activity executes, the onStart and onResume methods execute, placing the Activity in Resumed state, passing through Created and Started along the way.

Your Activities can reach the Resumed state in more than one way, not just when the app is launched. If your Activity is Paused or Stopped, it can be brought back into the foreground without the onCreate method being called again. If your app is brought back into the Resumed state from the Paused state, the Activity onResume method executes. If the app is brought back from Stopped state, onRestart executes, followed by onStart and onResume.

3. Moving Towards Destroyed

Step 1:

When your app is either exited or hidden, it moves from Resumed towards being destroyed. The onPause method takes your Activity from its normal running Resumed state to Paused. In onPause, you should stop any ongoing resource-hungry tasks such as animations, sensor handling, and broadcast receivers. If onPause executes, there is a good chance onStop is about to execute too since the user is possibly navigating away from your app. You can also use the method to save data, although in general this is best done in onStop.

As we mentioned above, your Activity can go back from Paused to Resumed via the onResume method. This means that you can use onResume to restore anything you stopped or released in onPause. However, bear in mind that onResume also executes in other cases, such as when the app is launched.

Step 2:

After onPause, onStop executes if the app enters Stopped state. From here, onRestart, onStart, and onResume can still take it back to Resumed. In onStop you should carry out any data saving operations you need, such as writing to databases. Make sure you release any resources your app is using in onStop to prevent any memory leaks if your app is destroyed.

The system saves certain data items when your app resumes after it is stopped, such as what is displayed in the Views. When an Activity moves from Stopped back to Resumed, the onRestart, onStart, and onResume methods all execute. However, onStart and onResume execute in other cases, such as when the app is launched. The onRestart method only executes when the app is coming back from Stopped state, so you can use it to reinstate anything you did in onStop.

Step 3:

If your app is about to completely finish, like with the instance of your current Activity removed from the system, onDestroy executes. Although this is the last method to execute before your Activity disappears completely, you should not typically use it to clean up. You should generally use onStop or onPause for this. One exception to this is where a background thread is running, in which case you should stop it in onDestroy.

After onDestroy executes, if the user navigates back to your app Activity the onCreate method executes again. In general, you can assume that onPause and onStop will execute before onDestroy. However, if you explicitly call the finish method to end an Activity, only onDestroy executes.

In many cases, you don't need to worry about most of the life cycle callbacks in your apps at all, since you can retain data using the onCreate method parameter. The Bundle parameter to your Activity onCreate method automatically saves View information as we mentioned above. However, you can also use the object to save additional data, such as variables you update as the user interacts with your app. To do this, you can implement the onSaveInstanceState method in an Activity class, writing key value pairs of data you can then retrieve in onCreate.

4.Conclusion

The various states and callback methods in Android Activities can be the source of much confusion when you start to learn how to develop apps for the platform. However, in most cases you only need to implement a minimal number of the methods to keep your apps functional and behaving the way users expect. In the next part of the series, we will look at some of the common Android classes you are likely to find useful in your first apps. After that we will look at the Android sample code, what you need to know about releasing your apps, and some other suggestions for future learning.

（1）Activity 与生命周期

介绍

Activity 生命周期并不仅仅在用户运行应用程序之后才开始生效，事实上它也影响着用户切出以及切回应用时得到的不同反馈。当我们开发一款应用时，首先需要牢记一点：用户会经常在执行过程中、在我们的应用与其它应用之间频繁切换。取决于用户的操作方式，同一款应用程序有时在前台运行、有时则在后台运行。大家必须保证自己的应用能够就会这类情况，并在此类切换过程中及时保存并恢复数据。再次提醒各位，这一过程对于某些特定应用程序略有不同——例如功能性组件。

1.回调方法

第一步 ：

为了控制 Activity 处于不同状态下时应用程序的运行方式，例如当用户切出或者切回应用，大家可以选择多种处理方法。这类方法也就是 Activity 生命周期回调方法。Android 系统会在我们的 Activity 进入某种特定状态后调用这些方法，从而通过一系列步骤确保我们的应用程序能够继续起效、不至于丢失数据而且在用户不与之交互时不会使用非必要性资源。每一种回调方法都会让我们的应用进入一种可能的状态。

如果大家之前曾经接触过 Java 应用程序的编程工作，那么应该已经发现 Android 应用程序的启动遵循另一种方式。与 Java 应用直接使用主方法不同，Android 在启动后会首先执行主 Activity 类中的 onCreate 方法。请记住，我们已经在清单中将该类指定为主启动 Activity。Activity 会首先回调 onCreate 方法，相当于重复用户启动应用程序后的流程。这时候 onCreate 方法会使应用程序进入 Created 状态。

开发者指南当中通过示意图以直观方式介绍了生命周期、回调方法以及状态的概念。其中 onResume 方法负责提供 Resumed 状态，这时我们的应用程序可以接受用户的直接操作。其它各类回调方法都以 onResume 为核心，即将应用程序引导至 Resumed 状态或者从该状态脱离、启动该状态或者将其停止。

对于大部分应用程序来说，我们只需要使用一部分回调方法，但最起码要用到 onCreate。虽然使用频率不高，但了解全部回调及状态的作用将帮助我们了解自己的应用程序在运行及停止运行时，Android 系统会受到怎样的影响。一般情况下，大家需要保证用户能够在任何操作过程切换出去之后、都能顺利恢复到之前的运行状态；如果他们通过导航选择前进或者后退，应用则需保存全部必要数据并释放不必要占用的硬件资源。

第二步 ：

我们的应用程序可能处于以下五种状态，分别为：Created、Started、Resumed、Paused 以及 Stopped。另有七种回调方法能够让应用进入或者脱离上述状态，它们分别是：onCreate、onStart、onRestart、onResume、onPause、onStop 以及 onDestroy。这些方法能够让我们的应用程序在可能的状态之间进行切换，而且某些情况下切换速度会很快。通常来说，大家可以认为自己的应用程序始终处于 resumed、paused 或者 stopped 这三种状态之下，因为其它状态都是暂时性的。

当我们的应用程序正处于运行当中且用户与之进行操作交互，这时的应用状态为 Resumed；当另一个 Activity 处于前台但仅仅使我们的应用被部分隐藏时，这时的应用状态为 Paused——在这种状态下用户无法再与应用进行交互。当我们的应用完全处于后台之下，而且用户既无法操作、也无法观看到它时，其状态即为 Stopped。在这种状态下 Activity 会保留之前的所有数据，但无法加以执行。

2.进入 Resumed 状态

如我们所知，主 Activity 会在应用程序启动时开始运行，onCreate 方法也将执行、从而让我们准备该类所需要的 Activity UI 以及全部数据条目。我们创建的大部分应用当中都包含不只一个 Activity，其它 Activity 会在用户与应用程序进行操作交互时启动。大家可以利用以下代码通过 Intent 类启动另一个非主 Activity：

这代表着应用程序包中另一个名为“About”的 Activity 类。大家可以通过选择自己的源码包而后选择“文件”、“新建”、“类”的方式在 Eclipse 当中创建一个新 Activity，而后将该 Android Activity 类选定为超级类。请记住，每一个 Activity 都必须在我们的应用程序清单当中列出。大家还可以利用 Intent 类实现不同 Activity 之间的数据转移。

当一个 Activity 处于运行当中时，onCreate 方法也在同时执行，因此除了把其它 Activity 类列入清单之外、大家也能够以与主 Activity 类似的方式在应用程序当中处理这些类。我们也可以为每个 Activity 创建一个布局文件，并通过设置让其使用与主 Activity 同样的技术机制。

在某个 Activity 的 onCreate 方法开始执行之后，onStart 与 onResume 两个方法也将开始执行， 从而使该 Activity 处于 Resumed 状态、并在后续执行过程中根据情况转换为 Created 以及 Started 状态。

我们的 Activity 可以通过不只一种方式进入 Resumed 状态，应用程序启动只是其中最基本的途径。如果 Activity 处于 Paused 或者 Stopped 状态，则应用程序切换至当前之后该 Activity 将直接进入前台运行模式，且无需重复调用 onCreate 方法。如果大家的应用从 Paused 状态切换回 Resumed 状态，则 Activity 的 onResume 方法将开始执行。如果该应用由 Stopped 状态切换回运行状态，则执行 onRestart 方法、而后依次为 onStart 与 onResume 方法。

3.进入 Destroyed 状态

第一步：

当我们的应用程序处于退出或者隐藏状态下，则 Resumed 就会转变为 Destroyed。这时候，onPause 方法会将应用的 Activity 由运行时的 Resumed 状态转换为 Paused 状态。在 onPause 当中，大家应当停止任何需要占用资源的任务，例如动画播放、传感器数据处理以及广播接收等等。如果 onPause 正在执行，那么 onStop 也可以开始执行，因为用户此时通常已经通过导航退出了我们的应用程序。大家还可以利用 onPause 方法进行数据保存——虽然通常来说数据保存工作由 onStop 方法来负责最为妥当。

正如我们之前曾经提到，大家的 Activity 能够通过 onResume 方法从 Paused 状态重新回归至 Resumed 状态。这意味着我们可以利用 onResume 来恢复任何我们之前在 onPause 当中停止或者发布过的内容。不过大家还需要记住一点，onResume 在其它情况下也会付诸执行，例如在应用程序启动时。

第二步 ：

在 onPause 之后，如果应用程序进入 Stopped 状态，那么 onStop 也将开始执行。在这种情况下，onRestart、onStart 以及 onResume 等方法仍然能够使应用程序重新回到 Resumed 状态。在 onStop 中，大家应当尽可能压缩只在必要数据的操作量，例如向数据库中写入内容。请大家确保在 onStop 当中囊括了所有应用程序所使用的资源，从而避免该应用在被彻底关闭之后导致内存溢出问题。

这套系统会在应用程序从 resumed 状态切换至 stopped 状态后保存特定数据，例如视图中需要显示的内容。当某个 Activity 从 Stopped 状态恢复到 Resumed 状态时，onRestart、onStart 以及 onResume 方法都会开始执行。不过 onStart 与 onResume 的执行情况有所不同——例如在应用程序启动之时。而 onRestart 方法只会在应用程序从 Stopped 状态恢复至前台之后才会执行，这样大家就能利用它来恢复任何保存在 onStop 当中的运行内容。

提示:当大家从一个 Activit 之下启动另一个 Activity 时，前者会进入 Stopped 状态。如果用户随后利用后退按钮再次由后者返回先前的 Activity 当，那么前者的 onRestart 方法就会开始执行。

第三步：

如果大家的应用程序即将彻底关闭，例如我们的当前 Activity 被从系统当中移除，则 onDestroy 方法会开始执行。尽管这是在我们的 Activity 完全消失之前执行的最后一个方法，大家仍然不应该简单地将所有内容一股脑清除。事实上，我们需要利用 onStop 或者 onPause 来处理结束工作。当然也有例外情况，如果应用程序的后台进程仍然处于运行状态，那么这时候大家应该在 onDestroy 当中将其停止。

在 onDestroy 执行之后，如果用户通过导航返回应用程序 Activity，则对应 onCreate 方法将再次被启动。一般情况下，大家可以假设 onPause 与 onStop 会在 onDestroy 之前执行。不过如果大家明确调用 finish 方法来结束一个 Activity，则只有 onDestroy 会被执行。

在多数情况下，我们并不需要为应用程序当中的生命周期回调问题投入过多精力，因为大家完全可以利用 onCreate 方法的参数实现数据保留效果。在 Activity onCreate 方法当中，Bundle 参数负责如前所述自动进行视图信息保存。不过大家也可以利用该对象保存更多数据内容，例如记录用户与应用程序之间的交互所产生的变量更新。要实现这一目标，大家可以在 Activity 类当中使用 onSaveInstanceState 方法，完成数据键值对的编写之后、我们就可能在 onCreate 当中将其恢复。

提示:当用户改变设备显示模式时，也就是在纵向及横向模式间进行切换，我们的 Activity 实际上会经历重新创建、onCreate 也会被再次执行。这一过程被我们称为配置变化。在这种情况下，系统会假设大家需要重新创建 Activity，例如大家在每种显示模式下使用不同的布局方案。不过在多数情况下，大家可能不希望系统照此办理。为了避免我们的 Activity 在显示模式转换时发生重新创建，大家可以从两种解决方式中作出选择：向清单内的 Activity 添加“android:configChanges”属性，或者调整我们的 Activity 结构、利用我们在配置变量时所保留的 Fragments。

4总结

当大家开始学习如何为 Android 平台开发应用程序时，Activity 当中所涉及的大量状态与回调方法可能会成为很多难题乃至混乱的根源。然而在大多数情况下，我们只需要采用最低数量的方法以确保自己的应用程序有能力提供用户所预期的功能与效果。在本系列教程的下一篇当中，我们将共同了解部分常用 Android 类，大家很可能会在自己的第一款应用当中与它们打交道。在此之后，我们将着眼于 Android 代码示例、需要了解的应用程序发布知识以及其它一些关于今后进一步学习的建议。

（2）Android SDK: App Structure

Introduction

This tutorial will primarily involve exploration, but we will begin development tasks in the next few tutorials, when we look at building user interfaces, responding to user interaction, and using Java to code application logic. We will focus on the project ingredients that you are most likely to interact with when you start to develop for Android, but you will find other elements already inside your application structure and many more are possible. We will not go into too much detail about any particular element in this tutorial, but will get to know some of the basic building blocks of an Android app.

1. Source

Step 1

Open Eclipse and expand the Package Explorer folder for the project we created. Inside the "src" folder you should see the package you named when you set the project up. Inside the package should be your Activity class file, which should also be open in the editor. The source folder holds all of the Java files you work on when you develop Android apps.

Each time you create a project you will create a package with your Java class files in it. An app may have more than one package in it and each package may hold multiple class files. The class files contain the processing code that presents your app to the user, responds to user interaction, and carries out any necessary processing. Essentially, the class files divide up the code involved in the application according to the Object Oriented conceptual model.

We will cover more about Java concepts and practices later in this series. For now, just understand that a Java application splits the various processing tasks between a number of objects. Each object is defined by a class declaration, which is typically a single file in an application, but which can also be nested inside another class file. An object is basically a chunk of code which carries out some part of the functionality involved in the app. The code in a class file can refer to the other classes in the application or in any package within the application.

When you start developing apps, you need to add Java classes to your package(s) in the source folder. A typical Android app that presents a user interface to the user will have at least one Activity file in it, with extra Activity classes for each screen in the app. There are other types of app such as those involving widgets and services, which can adopt a different structure. It's best to learn about the Activity-focused UI type of app to begin with and learn about the others later.

Step 2

Have a look at the Activity class file in your new app. We will explore more of the Activity code later on in this series, so don't worry too much about the details. This is your app's main Activity, which starts when the app launches. Your app may then launch other Activities on user interaction. When you created your project, Eclipse sets the app up to use this as the main class. It is listed as the main Activity in the project Manifest, which we will look at soon.

Inside the main Activity class, you will see the onCreate method, which contains the code that will execute when the Activity is created, i.e. when the app is launched. Inside the method you will see the following line of code:

setContentView(R.layout.activity\_main);

This line specifies the layout file we created when we started the project, telling Android to use it as content view. This means that whatever is in the layout file will be what users see when this Activity is on the screen.

We will look more at this later but for the moment notice the "R.layout.activity\_main" syntax. This is how your Java code refers to the resources in the app. We will use similar syntax to refer to resources by their ID value, as well as referring to other types of resources such as images and data values. The "R" represents the app resources and what follows specifies the item type, in this case a layout, stored inside the "res/layout" directory. The resource is finally identified using its name - in the case of the layout this is the filename. The syntax is therefore "R.type.name". You will get used to this when we start to code.

Later in this series we will add code to the Activity class file to handle user interaction. Expand your app "res" folder now. Inside it you will see a number of sub-folders. These are the folders Eclipse and ADT create by default when you start a new Android project, but there are a number of other possible directories you can also add for different types of resources.

2. Layout Resources

As we have already seen, the layout file formed when the project was created appears in the "res/layout" folder. If an app has multiple Activity screens, it will typically have a layout file for each. You may also use layout files for individual UI items. When you create the class file for an Activity, you set the layout usingsetContentView as we saw above. You can alternatively define a layout in Java code, in which case it is built dynamically when the app executes. However, the advantage to using XML is that you can see a visual representation of the layout while you design it.

Inside the main layout file for your app, which should be open in the editor, you will see XML structures. Don't worry if you have no XML experience, we will run over the basics later on in this series. For now, just understand that XML is a markup language, similar to HTML if you have tried Web development before. XML files model data within a tree structure. Typically a layout file has a root layout element modeling a particular type of layout, with child elements inside it for the UI items such as buttons, images, and text.

3. Drawable Resources

You will see multiple folders in the resources directory with "drawable" in the name. These store the image files your app uses. These image files can be digital image files you prepare outside Eclipse, with formats such as PNG or JPEG. Alternatively, you can define certain drawables using XML code to describe shapes, color, and appearance. Once you have a file in your drawable folders you can refer to it in the app layout files or in Java code. This allows you to build images into your app's UI.

The resource directory includes drawable folders for each density bucket. The density buckets are generalized categories for the the different screen densities on devices running Android. The generalized categories are for low, medium, high, extra high, and extra extra high density. Using these allows you to simplify the process of supporting multiple screen densities by catering for each of these categories. This means that when you include image files in your projects, you can include versions of them in each density folder, tailoring the images to the densities in each case.

4. Data Resources

In your "res" directory you will see some folders with "values" in the title. These are for data values you wish to use within your app. Such values can include text strings and numbers. The values folders contain XML files in which one or more values are listed. Each listing includes a name and the value in question. Other files in the app, such as Java class or layout files, can refer to the values using their names. A typical use for such a value would be to store a text string to display within a UI element such as a button.

The different values files in the app are designed to allow you to tailor values to particular screen sizes and API levels. If the same value can be used across devices, it can be saved in the plain "values" folder.

5. The Manifest

If you look in the main folder for your app, you will see the project Manifest file. Open it in the editor by double-clicking it. You will see a graphical interface to its content. Click the "AndroidManifest.xml" tab at the bottom of the editor window to see the XML code. This file defines multiple aspects of the app as a whole. Eclipse and ADT build certain elements into the Manifest when you create the app, basing these on the settings you chose during project creation. You can add other elements to the Manifest manually. For example, if you add other Activities to your app.

We will run through some of the main elements to understand the Manifest, but there are many other elements you can include. Listed in the Manifest for your new app project you will see the uses-sdk element, in which we indicate minimum and target API levels. The application element contains attributes indicating the launcher and app name. Inside the application element is anactivity element, listed as the main Activity to launch when the app runs via theintent-filter element. When you add new Activities to an app you will include a new activity element for each.

Other elements you may need to add to the Manifest include the uses-permission element in which you list permissions the app requires - the user sees a list of these before installing the app. Permissions include actions such as fetching data over the Internet, writing to storage, or accessing other features of the device such as the camera. The Manifest also lists data regarding which devices the app supports, as well as lists other app components such as background services.

6. Other Files

So far we have covered the main aspects of an Android app project structure that you need to know for your first few apps. We will work with these files as we learn the skills involved in Android development. There are several other files and directories in the project as you can see in Eclipse, but for the most part you can ignore them for now.

As we saw above, you can refer to resources using the "R." syntax. Eclipse and the ADT manage the system that refers to the resources in your app from Java. When you add or edit the resources in your project, Eclipse writes to the "R.java" file, which in turn allows you to refer to the resources using "R.". When you start working in your Java files, you will see that Eclipse will prompt you with suggestions when you refer to R, making it easier to manage the resources in your app.

The "R.java" file is stored in the "gen" folder. Do not attempt to edit this file directly, it automatically generates when you edit the resources in your project. The system manages this process by allocating a unique integer ID to each resource in your app.

Conclusion

In this tutorial we explored the basics of an Android project structure. Feel free to spend more time exploring the other files and folders in your project to get to know its overall structure. In the following sections of this series we will build user interface elements and handle user interaction with our app. We will also look at some of the essential features of Java programming to understand it in order to get the best out of our Android development project.

（2）Android 应用程序结构

介绍

本教程将主要以探索与了解为主要目的，但后续的系列文章则将进一步带大家深入学习如何创建用户界面、响应用户交互操作以及利用 Java 编排应用逻辑。我们将专注于大家刚刚开始接触 Android 开发时最常遇到的项目内容，但也会同时涉及一部分已经存在于应用结构当中的其它一些元素。在今天的文章中，我们不会对这些额外元素进行深入探讨。总而言之，了解关于 Android 应用的基础创建知识，这就是我们今天要完成的教学任务。

1. 源

第一步

打开 Eclipse 并在 Package Explorer 当中查找我们已经创建完成的项目。在“src”文件夹里，大家应该会看到设置项目所命名的项目包。包中应该包含我们的 Activity 类文件，这也是要在编辑器中打开的内容。源文件夹保存着我们在开发 Android 应用程序时所要用到的全部 Java 文件。

每当我们创建一个项目时，都会创建一个用于容纳各 Java 类文件的包。一款应用程序可能拥有不止一个包，而且每个包当中也可能容纳着多个类文件。这些类文件中的处理代码能够将我们的应用呈现给用户、响应用户的交互操作并执行任何的必要处理。从实质上看，类文件是在根据面向对象概念模型划分与应用程序相关的代码。

我们将在后续文章中进一步讨论关于 Java 的概念以及对应实践。在今天的教程内，大家只需理解一个 Java 应用会将各类处理任务拆分成一定数量的对象。每个对象都由一个类声明来定义，这在应用程序中通常是一个独立的文件，不过也可以被嵌套在其它类文件当中。一个对象基本上就是一大段代码，其中承载着与应用程序相关的某项功能的一部分。类文件中的代码能够引用应用程序中的其它类或者应用程序中的其它包。

在大家着手进行应用程序开发时，首先需要向源文件夹中的包中添加 Java 类。一个向用户提供用户界面的典型 Android 应用将拥有至少一个 Activity 文件，应用中的不同屏幕显示内容还要用到更多 Activity 类。其它一些类型的应用，例如工具程序或者服务，则采用不同的结构。大家最好首先关注 Activity UI 这种类型的应用程序，并在熟练掌握之后再接触其它应用类型。

第二步

现在查看新应用中的 Activity 类文件。我们会在本系列教程的后续文章中进一步探讨 Activity 代码，因此目前大家不用太过关注细节。今天我们主要面向应用中的主 Activity，它会在应用启动后同时开始生效。大家的应用也可能会启动其它一些用于用户交互的 Activity。在我们创建自己的项目时，Eclipse 会对应用进行设置并将主 Activity 作为主类——它在项目清单当中也将被作为主 Activity 进行显示，我们稍后会看到。

在主 Activity 类当中，大家会看到 onCreate 方法，其中包含的代码将在 Activity 被创建——也就是应用程序启动时开始执行。在该方法中，大家会看到以下代码行：

setContentView(R.layout.activity\_main);

在我们启动项目之后，这一行的内容用于指定我们所创建的布局文件，告诉 Android 将其作为内容视图。这意味着无论布局文件中包含什么样的内容，都将在该 Activity 显示在屏幕上时呈现给用户。

我们将在稍后进一步探讨相关话题，目前暂时需要将注意力集中在“R.layout.activity\_main”语法上。这就是我们的 Java 代码引用应用程序资源的方式。我们将利用类似的语法通过资源的 ID 值对其进行引用，例如图片及数据值等资源也可以通过这种方式实现引用。其中的“R”代表应用资源，后面的部分则用于指定保存在“res/layout”目录下的条目类型——在这里就是布局。这些资源最终要根据其名称进行识别——对于示例中的布局，使用的就是文件名。由此推断，我们要使用的语法就成了“R.type.name”。在我们开始编程之后，各位就会开始使用该语法。

在本系列的后续文章中，我们将向 Activity 类文件中添加代码以实现用户交互。现在打开应用中的“res”文件夹，大家会在其中找到多个子文件夹。这些文件夹是由 Eclipse 与 ADT 在我们启用新 Android 项目后默认创建而成的，不过我们可能还需要为不同类型的资源添加其它一些目录。

2. 布局资源

正如我们已经看到，项目创建后所生成的布局文件会保存在“res/layout”文件夹中。如果某款应用拥有多个 Activity 屏幕，那么一般会为每个屏幕保留一个独立的布局文件。大家可能还会将布局文件用于个别 UI 条目。当大家为 Activity 创建类文件时，需要如上所述利用 setContentView 进行布局设置。除此之外，大家也可以通过 Java 代码进行布局设置——这算是种备选方案。在我们的示例中，布局设置是在应用执行时动态生成的。不过利用 XML 的优势在于，我们可以在界面设计工作中直观感受布局方案的视觉效果。

在应用程序的主布局文件当中（现在应该已经用编辑器打开了），大家会看到 XML 结构。如果各位之前没有接触过 XML 也不必担心，我们会在后续文章中进一步讨论这些基础知识。就目前来说，大家只需了解：XML 是一种标记语言，类似于 HTML——如果之前接触过 Web 开发的话。XML 文件利用树状结构作为数据模型。通常来说，一个布局文件拥有一个根布局元素，并将其作为特定布局类型模型——其中所包含的用于 UI 条目的子元素则包括按钮、图片及文本等。

3. 可绘制资源

大家在资源目录下应该会看到多个在名称中包含“drawable”字样的文件夹，这些文件夹用于保存应用程序所使用的图片文件。这些图片文件可以是我们在 Eclipse 之外所准备的数字图片文件，格式包括 PNG 或者 JPEG 等。或者，大家也可以通过 XML 代码来描述形状、颜色以及外观，从而定义特定可绘制资源。一旦我们在 drawable 文件夹中创建了文件，就可以在应用布局文件或者 Java 代码中进行引用。这样一来，之前准备好的视觉元素就能用于应用 UI 了。

资源目录中会保留针对每一种密度桶的 drawable 文件夹。这些密度桶是各类运行 Android 系统的设备在像素密度方面的通用型分类依据。具体类别分为低、中、高、超高与超超高密度四种。只需从对应类型中作出选择，我们就可以在密度桶的帮助下轻松简化多屏幕密度的支持过程。这意味着当我们在项目中包含图片文件时，可以将其放置在不同密度的文件夹当中，并通过裁剪提供满足各种密度方案的版本。

4. 数据资源

在“res”目录中，我们会看到一些标题中带有“values”字样的文件夹。这些文件夹用于容纳大家希望在应用程序中所使用的数据值。这些值可以包含文本字符串以及数字。包含 XML 文件的值文件夹会列出其中的一项或者多项值。每份列表都包含一个名称以及内容中的值。应用中的其它文件，例如 Java 类或者布局文件，能够通过这些名称为引用这些值。在典型用例中，我们能够需要通过这些保存在文本字符串的值在 UI 元素当中显示内容——例如按钮。

5. 清单

在查看应用程序中的主文件夹时，大家一定会发现项目的清单文件。通过双击即可利用编辑器将其打开。接下来，我们会看到一个显示其内容的图形界面。点击编辑器窗口底部的“AndroidManifest.xml”标签来查看其 XML 代码。这个文件将应用程序的各个方面定义成统一整体。Eclipse 与 ADT 会在我们创建应用的同时，在清单中创建特定元素，具体创建方式取决于大家在项目创建过程中的设置。大家可以手动向清单中添加其它元素，例如添加其它 Activity。

我们将运行其中的一部分主元素，旨在理解清单的作用，不过还有其它多种元素可以被包含其中。在清单中所列举的新应用项目元素当中，我们将看到 uses-sdk 元素，我们利用它表示最小及目标 API 级别。Application 元素中包含指向启动机制与应用程序名称的属性。在 application 元素中还存在着一个 activity 元素，会在应用程序开始运行时通过 intent-filter 元素作为主 Activity 启动。当我们向应用中添加新的 Activity 时，则会为每个相关元素添加新的 activity 元素。

大家可能还需要向清单中添加其它元素，其中包括 uses-permission 元素，用于罗列应用所要求的权限——用户会在安装应用之前观看到该列表。权限中包含多种操作条目，例如通过互联网获取数据、写入存储或者访问设备上的其它功能——如相机。清单还会列举应用程序所能支持的设备类型以及其它一些应用程序组件（例如后台服务）。

6. 其它文件

讲到这里，我们已经谈到了大家需要了解的 Android 应用程序项目结构中的各大主要方面。随着对 Android 开发的学习，大家将在今后经常与这些内容打交道。通过 Eclipse，我们还会看到项目中包含的其它一些文件及目录，不过就目前来说基本都可以直接忽略。

正如在前面看到的，大家可以利用“R.”语法实现资源引用。Eclipse 以及管理系统的 ADT 都会引用应用中来自 Java 的资源。当大家在项目中对这些资源进行添加或者编辑时，Eclipse 会将对应内容写入“R.java”文件，从而帮助我们利用“R.”进行资源引用。当大家开始处理自己的 Java 文件，会在引用时看到 Eclipse 弹出的提示信息——这种机制能简化对应用资源的管理工作。“R.java”文件被保存在“gen”文件夹中。请注意：千万不要尝试直接编辑这个文件，它会在我们编辑项目资源时自动生成。系统会通过为应用中的每项资源分配惟一整数 ID 的形式管理这一过程。

总结

在今天的文章中，我们了解了关于 Android 项目结构的基础知识。大家可以再花点时间随意查看项目中的其它文件及文件夹，借此了解项目的整体结构。在接下来的后续教程中，我们将在应用中创建用户界面元素并处理用户交互操作。我们还会探讨关于 Java 编程的基本特性，借此进一步提升自己对 Android 开发项目的理解。