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 4 章 Android 生命周期

本章简介

经过上一章的学习，主要了解了 Android 应用程序设计的基础知识，对 Android 程序的开发有了一定的了解。在此基础上，本章将对 Android 系统的进程优先级的变化方式、Android 系统的 4 大基本组件、Activity 的生命周期中各个状态的变化关系、Android 应用程序的调试方法和工具进行学习。

4.1 程序生命周期

所谓的应用程序生命周期就是应用程序进程从创建到消亡的整个过程。在 Android 中，多数情况下每个程序都是在各自独立的 Linux 进程中运行的。当一个程序或其某些部分被请求时，它的进程就“出生”了；当这个程序没有必要再运行下去且系统需要回收这个进程的内存用于其他程序时，这个进程就“死亡”了。可以看出，Android 程序的生命周期是由系统控制而非程序自身直接控制。这和编写桌面应用程序时的思维有一些不同，一个桌面应用程序的进程也是在其他进程或用户请求时被创建，但是往往是在程序自身收到关闭请求后执行一个特定的动作（如从 main 方法中 return）而导致进程结束的。

简而言之，程序的生命周期是在 Android 系统中进程从启动到终止的所有阶段，也就是 Android 程序启动到停止的全过程，程序的生命周期是由 Android 系统进行调度和控制的。

但是，一个不容忽视的问题就是，手机的内存是有限的，随着打开的应用程序数量的增多，随之而来的可能会是应用程序响应时间过长或者系统假死的糟糕情况。所以，若将 Android 应用程序生命周期交由系统处理的话，那么在系统内存不足的情况下，便由 Android 系统舍车保帅，选择性地来终止一些重要性较次的应用程序，以便回收内存供更重要的应用程序使用。

那么，系统是根据一个什么样的重要性标准来终止 Android 应用程序的呢?

Android 根据应用程序的组件及组件当前运行状态将所有的进程按重要性程度从高到低划分了五个优先级：前台进程、可见进程、服务进程、后台进程、空进程。

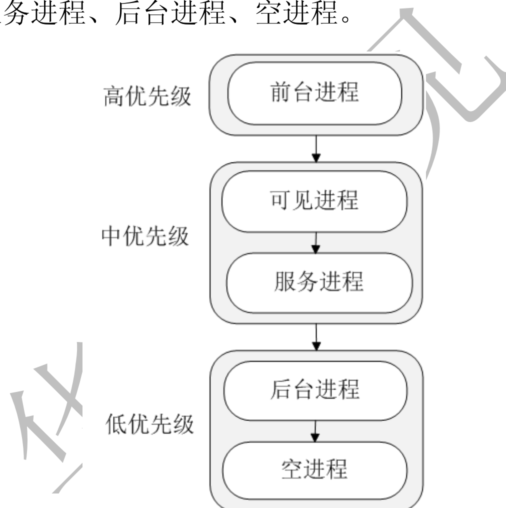


图 4-1 Android 系统进程优先级

以下就按优先级由高到低的顺序介绍 Android 系统中的进程。

1. 前台进程

前台进程是显示在屏幕最前端并与用户正在交互的进程，是 Android 系统中最重要的进程，包含以下 4 种情况。

- 进程中的 Activity 正在与用户进行交互。
- 进程服务被 Activity 调用，而且这个 Activity 正在与用户进行交互。
- 进程服务正在执行声明周期中的回调方法，如 onCreate()、onStart()或 onDestroy()
- 进程的 BroadcastReceiver 正在执行 onReceive()方法。

Android 系统在多个前台进程同时运行时，可能会出现资源不足的情况，此时会清除部分前台进程，保证主要的用户界面能够及时响应。

2. 可见进程

可见进程指部分程序界面能够被用户看见，却不在前台与用户交互，不响应界面事件（其 onPause()方法已被调用）的进程。如果一个进程包含服务，且这个服务正在被用户可见的 Activity 调用，此进程同样被视为可见进程。

Android 系统一般存在少量的可见进程，只有在特殊的情况下，Android 系统才会为保证前台进程的资源而清除可见进程。

3. 服务进程

服务进程是指包含由 startService()方法启动服务的进程。它有以下特性：没有用户界面；在后台长期运行。例如，后台 MP3 播放器或后台上传下载数据的网络服务。

Android 系统除非不能保证前台进程或可见进程所必要的资源，否则不强行清除服务进程。

4. 后台进程

后台进程是指不包含任何已经启动的服务，而且没有任何用户可见的 Activity 的进程。这些进程不直接影响用户的体验。

Android 系统中一般存在数量较多的后台进程，因此这些进程会被保存在一个列表中，以保证在系统资源紧张时，系统将优先清除用户较长时间没有见到的后台进程。

5. 空进程

空进程是不包含任何活跃组件的进程。一般保留这些进程，是为了将其作为一个缓存，在它所属的应用组件下一次需要时，缩短启动的时间。

空进程在系统资源紧张时会被首先清除，但为了提高 Android 系统应用程序的启动速度，Android 系统会将空进程保存在系统内存中，在用户重新启动该程序时，空进程会被重新使用。

问：除了以上的优先级外，还有其他因素决定进程的优先级吗？

答：

- 进程的优先级取决于所有组件中的优先级最高的部分。
- 进程的优先级会根据与其他进程的依赖关系而变化。

华清远见

4.2 Android 组件

组件是可以调用的基本功能模块。Android 应用程序就是由组件组成的，Android 系统有 4 个重要的组件，分别是 Activity、Service、BroadcaseReceiver 和 ContentProvider。

Activity 是 Android 程序的呈现层，显示可视化的用户界面，并接收与用户交互所产生的界面事件。在界面上的呈现形式就是全屏窗体、非全屏悬浮窗体的对话框，与在桌面系统上的独立事业，如办公应用等类似。Activities 是可执行的代码块，由用户或者操作系统来进行初始实例化，并在他们被需求时致以运行。Activities 可以与用户、请求数据或者其他 Activity、Service 的服务通过 query 或 Intent 进行交互。大部分为 Android 编写的可执行代码将以 Activity 的形式执行。对于一个 Android 应用程序来说，可以包含一个或多个 Activity，一般在程序启动后会呈现一个 Activity，用于提示用户程序已经正常启动。当它不积极运行时，Activity 可以被操作系统终止以节省内存。

Service 用于没有用户界面，但需要长时间在后台运行的应用。它类似于桌面应用或者服务器操作系统上的服务或守护进程。Service 是在后台运行的可执行的代码块，从它被初始化一直运行到该程序关闭。一个 Service 的典型的例子是一个 MP3 播放器，尽管用户已经使用其他应用程序，但仍然需要持续播放文件。你的应用程序可能需要在没有用户界面的情况下一直执行 Service 来实现后台任务。

Broadcast 和 Intent Receivers 对从其他的应用程序的服务请求做出一个全系统广播的响应，这些广播响应可能来自于 Android 系统本身或者是任何在其系统上运行的程序。BroadcaseReceiver 是用来接受并响应广播消息的组件。它不包含任何用户界面，但可以通过启动 Activity 或者 Notification 通知用户接收到重要信息。

问：Notification 如何提示用户？

答：闪动背景灯、振动设备、发出声音或在状态栏上放置一个持久的图标。

Activity 或 Service 通过执行一个 IntentReceiver 为其他应用程序提供了访问其功能的功能。Intent Receiver 是一段可执行代码块，对其他 Activity 的数据或服务请求做出响应。请求的 Activity（客户端）生成一个 Intent，把其添加至 Android Framework 中，来指出哪些应用程序（目标程序）接收并对其做出响应。Intent 是 Android 的主要构成元素之一，它从现有的应用程序中创造新的应用程序。Intent 实现了应用程序和其他的应用程序和服务交换所需信息的功能。

ContentProvider 是 Android 系统提供的一种标准的共享数据的机制，应用程序可以通过 ContentProvider 访问其他应用程序的私有数据（私有数据可以是存储在文件系统中的文件，也可以是 SQLite 中的数据库）。Android 系统内部也提供一些内置的 ContentProvider，能够为应用程序提供重要的数据信息。

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

4.3 Activity 生命周期

Activity 生命周期指 Activity 从启动到销毁的过程。Activity 表现为 4 种状态，分别是活动状态、暂停状态、停止状态和非活动状态。

- 活动状态，Activity 在用户界面中处于最上层，完全能被用户看到，能够与用户进行交互。
- 暂停状态，Activity 在界面上被部分遮挡，该 Activity 不再处于用户界面的最上层，且不能够与用户进行交互；或者屏幕被锁定。
- 停止状态，Activity 在界面上完全不能被用户看到，也就是说这个 Activity 被其他 Activity 全部遮挡。
- 非活动状态，不在以上 3 种状态中的 Activity 则处于非活动状态。

这四种状态是可以相互转换的，转换关系图如图 4-2 所示。

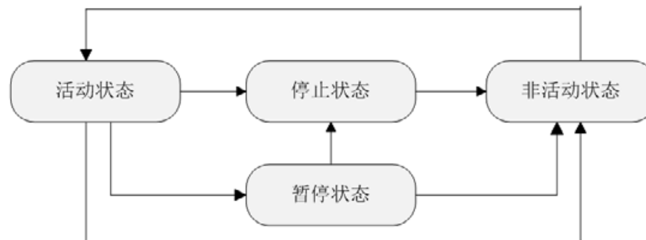


图 4-2 Activity 的 4 种状态的转换关系图

图 4-3 解释了状态之间转化的可能路径。其中着色的椭圆表示活动的主要状态，矩形表示当活动在状态之间转换时会被调用的回调方法。

Android 调用以下的事件回调方法通知 Activity 从某一状态转变到另一状态。

代码清单 4-1 事件的回调方法

```
public class MyActivity extends Activity {
    protected void onCreate(Bundle savedInstanceState);
    protected void onStart();
    protected void onRestart();
    protected void onResume();
    protected void onPause();
    protected void onStop();
    protected void onDestroy();
}
```

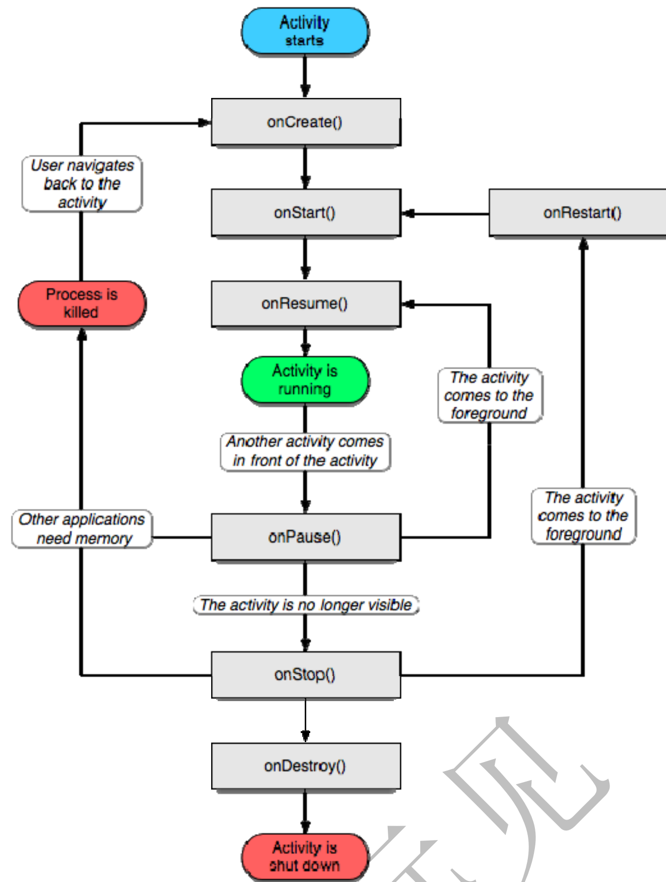


图 4-3 Activity 活动周期

表 4-1 对各个事件回调方法做出说明。

表 4-1 Activity 生命周期的事件回调方法

方法	是否可终止	说明
onCreate()	否	Activity 启动后第一个被调用的方法，常用来进行 Activity 的初始化，例如创建 View、绑定数据或恢复信息等
onStart()	否	当 Activity 显示在屏幕上时，该方法被调用
onRestart()	否	当 Activity 从停止状态进入活动状态前，调用该方法
onResume()	否	当 Activity 能够与用户交互，接收用户输入时，该方法被调用。此时的 Activity 位于 Activity 栈的栈顶
onPause()	是	当 Activity 进入暂停状态时，该方法被调用。一般用来保存持久的数据或释放占用的资源
onStop()	是	当 Activity 进入停止状态时，该方法被调用
onDestroy()	是	在 Activity 被终止前，即进入非活动状态前，该方法被调用
onSaveInstanceState()	否	Android 系统因资源不足终止 Activity 前调用该方法，用以保存 Activity 的状态信息，供 onRestoreInstanceState()或 onCreate()恢复之用
onRestoreInstanceState()	否	恢复 onSaveInstanceState()保存的 Activity 状态信息，在 onStart()和 onResume ()之间被调用

Activity 事件回调方法的调用顺序，如图 4-4 所示。

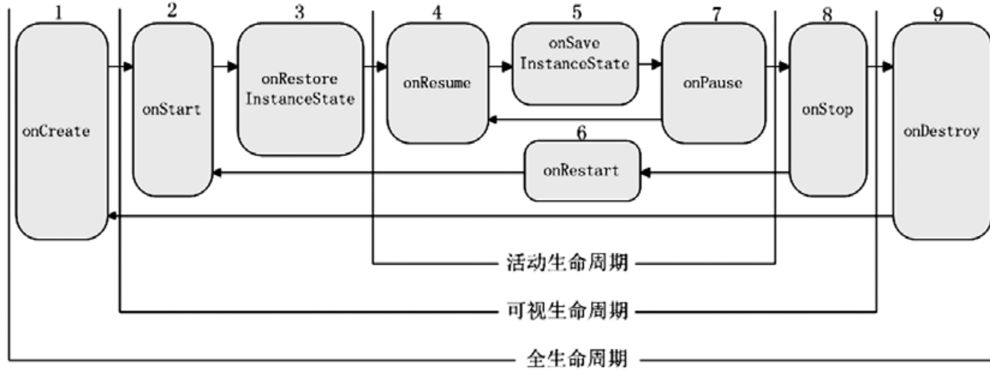


图 4-4 Activity 事件回调方法的调用顺序

Activity 的生命周期可分为全生命周期、可视生命周期和活动生命周期。每个生命周期中包含不同的事件回调方法。

4.3.1 全生命周期

全生命周期是从 Activity 建立到销毁的全部过程，始于 onCreate()，结束于 onDestroy()。

使用者通常在 onCreate()中初始化用户界面，分配引用类变量，绑定数据控件，并创建服务和线程等 Activity 所能使用的全局资源和状态，并在 onDestroy()中释放这些资源，并确保所有外部连接被关闭，例如，网络或数据库的联系等；在一些极端的情况下，Android 系统会不调用 onDestroy()方法，而直接终止进程。

为了避免创造短期对象和增加垃圾收集的时间，以致对用户体验产生直接影响。如果你的 Activity 需要创建一些对象的话，最好在 onCreate 方法中创建，因为它在一个 Actvity 的完整生命周期中仅调用一次。

4.3.2 可视生命周期

可视生命周期是 Activity 在界面上从可见到不可见的过程，开始于 onStart()，结束于 onStop()。

- onStart()一般用来初始化或启动与更新界面相关的资源。
- onStop()一般用来暂停或停止一切与更新用户界面相关的线程、计时器和服务。
- onRestart()方法在 onSart()前被调用，用来在 Activity 从不可见变为可见的过程中，进行一些特定的处理过程。
- onStart()和 onStop()会被多次调用。
- onStart()和 onStop()也经常被用来注册和注销 BroadcastReceiver 或者传感器。

在 onStart()和 onStop()这两个方法中间，Actvity 对用户将会是可见的，尽管它可能部分被遮挡着。在一个 Activity 完整的生命周期中可能会经过几个 Activity 可见的生命周期，因为 Activity 可能会经常在前台和后台之间切换。在极端情况下，系统将销毁掉一个 Activity 即使它在可见状态并且不调用 onStop 方法。

4.3.3 活动生命周期

活动生命周期是 Activity 在屏幕的最上层，并能够与用户交互的阶段，开始于 onResume()，结束于 onPause()。在 Activity 的状态变换过程中 onResume()和 onPause()经常被调用，因此这两个方法中应使用更为简单、高效的代码。

- onPause()是第一个被标识为“可终止”的方法。
- 在 onPause()返回后，onStop()和 onDestroy()随时能被 Android 系统调用。
- onPause()常用来保存持久数据，如界面上用户的输入信息等。

华清远见

当系统而不是用户关闭一个活动来节省内存时，用户可能希望返回到活动且是它之前的状态。为了获得活动被关闭之前的状态，可以执行活动的 onSaveInstanceState()方法。Android 在活动容易被销毁前调用这个方法，也就是调用 onPause()之前。该方法的参数是一个 Bundle 对象，这个对象可以名值对记录活动的动态状态。当活动再次启动时，Bundle 同时被传递到 onCreate()和调用 onCreate()之后的方法 onRestoreInstanceState()。

因为 onSaveInstanceState()方法不总被调用，你应该仅使用 onSaveInstanceState()来记录活动的临时状态，而不是持久的数据。应该使用 onPause()来存储持久数据。

问：onPause()和 onSaveInstanceState()这两个函数都可以用来保存界面的用户输入数据，它们有什么区别呢？

答：

- onPause()一般用于保存持久性数据，并将数据保存在存储设备上的文件系统或数据库系统中的。
- onSaveInstanceState()主要用来保存动态的状态信息，信息一般保存在 Bundle 中。
  - Bundle 是能够保存多种格式数据的对象。
  - onSaveInstanceState()保存在 Bundle 中的数据，系统在调用 onRestoreInstanceState()和 onCreate()时，会同样利用 Bundle 将数据传递给函数。

当一个活动启动另一个活动时，这两个活动都经历生命周期转换。一个暂停或是停止，然而被启动的活动则启动。有时，这些活动可能需要协调。当这两个活动在同一个进程中，生命周期的回调顺序是明确界定的：调用当前活动的 onPause()方法；然后，按序调用启动活动的 onCreate()、onStart()、onResume()方法；之后，如果该活动不需再在屏幕上可见，则调用它的 onStop()方法。下面我们就来详细学习一下关于 Android 如何管理多个 Activity。

（1）Android 用 Activity Stack 来管理多个 Activity，因此，同一时刻只会有最顶上的 Activity 是处于 active 或者 running 状态。其他的 Activity 都被压在下面。

（2）如果非活动的 Activity 仍是可见的（如果上面压着的是一个非全屏的 Activity 或透明的 Activity），它是处于 paused 状态的。在系统内存不足的情况下，paused 状态的 Activity 是有可能被系统销毁掉的。

注意

因为 Android 应用程序的生存期并不是由应用本身直接控制的，而是由 Android 系统平台进行管理的，所以，对于开发者而言，需要了解不同的组件 Activity、Service 和 IntentReceiver 的生命，切记：如果组件的选择不当，系统很有可能会关闭一个正在进行重要工作的进程。

华清远见

4.4 Activity 启动模式

Activity 作为 Android 中重要一环，它有 4 种不同的启动模式，类似于 C 语言中的局部变量、全局变量及静态变量等。这 4 种启动模式如下。

- standard：标准模式，调用 startActivity()方法就会产生一个新的实例。
- singleTop：检查是否已经存在了一个实例位于 Activity Stack 的顶部，如果存在就不产生新的实例，反之则调用 Activity 的 newInstance()方法产生一个新实例。
- singleTask：在一个新的 Task 中产生这个实例，以后每次调用都会使用此实例，而避免产生新的实例。
- singleInstance：这个基本上跟 singleTask 一样，只是有一点不同，那就是在这个模式下的 Activity 实例所处的 Task 中，只能有这一个 Activity 实例，而不能有其他的实例。

这些启动模式在 Android 清单文件 AndroidManifest.xml 中，通过<activity>中的 launchMode 属性进行设置，如代码清单 4-2 所示。

代码清单 4-2　AndroidManifest.xml

```
<activity android:name=".Activity2"
      android:launchMode="singleTask"></activity>
```

也可以在 Eclipse ADT 图形界面中编辑，如图 4-5 所示。

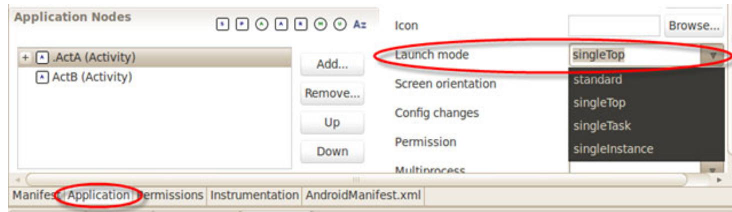


图 4-5 设置 Activity 启动模式

下面通过一个简单的例子——LaunchMode_Test 来对四种启动模式进行简要分析。在该例中涉及 Fx_Main、Activity2 及 Activity3 三个 Activity。

下面介绍一下例子中涉及的三个 Activity 及其界面。

首先是 Fx_Main，其界面如图 4-6 所示。

图 4-6 Fx_Main 的界面

在图 4-6 所示的界面中，单击“跳转到 AC2”按钮之后，跳转至 Activity2，具体代码如代码清单 4-3 所示。

代码清单 4-3 Fx_Main.Activity

```
import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.TextView;

public class Fx_Main extends Activity {
    /** Called when the activity is first created. */
  private Button b;
  @Override
protected void onCreate(Bundle savedInstanceState) {
  // TODO Auto-generated method stub
  super.onCreate(savedInstanceState);
  setContentView(R.layout.main);
  TextView tv=(TextView)findViewById(R.id.TextView01);
  tv.setText("Main---->"+getTaskId());
  Log.i("System.out", "Main---->"+this.toString()+"Task ID---->"+getTaskId());
  b=(Button)findViewById(R.id.Button01);
  b.setOnClickListener(new OnClickListener(){

        @Override
        public void onClick(View v) {
              // TODO Auto-generated method stub
              Intent i=new Intent(Fx_Main.this,Activity2.class);
```

```
            startActivity(i);
        }});
    }
    @Override
    protected void onDestroy() {
        // TODO Auto-generated method stub
        super.onDestroy();
        Log.i("System.out", "Fx_Main--->Destory");
    }
}
```

其次是 Activity2，其界面如图 4-7 所示。

图 4-7 Activity2 的界面

在该界面中，单击“跳回到 Main”按钮，则跳转至 Fx_Main，而单击“跳到本页面”则仍显示 Activity2 的界面，单击“跳到 AC3”则跳转到 Activity3 的界面，具体代码如代码清单 4-4 所示。

代码清单 4-4 Activity2.Activity

```
import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.TextView;

public class Activity2 extends Activity {
private Button b;
private Button b2;
private Button b3;
 @Override
protected void onCreate(Bundle savedInstanceState) {
 // TODO Auto-generated method stub
 super.onCreate(savedInstanceState);
 setContentView(R.layout.activity2);
 b=(Button)findViewById(R.id.Button02);
 b2=(Button)findViewById(R.id.Button03);
 b3=(Button)findViewById(R.id.Button04);
 TextView tv=(TextView)findViewById(R.id.TextView02);
 tv.setText("Ac2---->"+getTaskId());
 Log.i("System.out", "Ac2---->"+this.toString()+"Task ID---->"+getTaskId());
 b.setOnClickListener(new OnClickListener(){

      @Override
      public void onClick(View v) {
```

```
            // TODO Auto-generated method stub
            Intent i=new Intent(Activity2.this,Fx_Main.class);
            startActivity(i);
        }});
    b2.setOnClickListener(new OnClickListener(){

        @Override
        public void onClick(View v) {
            // TODO Auto-generated method stub
            Intent i=new Intent(Activity2.this,Activity2.class);
            startActivity(i);
        }});
    b3.setOnClickListener(new OnClickListener(){

        @Override
        public void onClick(View v) {
            // TODO Auto-generated method stub
            Intent i=new Intent(Activity2.this,Activity3.class);
            startActivity(i);
        }});
    }
    @Override
        protected void onDestroy() {
            // TODO Auto-generated method stub
            super.onDestroy();
            Log.i("System.out", "Ac2--->destory");
    }
}
```

最后是 Activity3，其界面如图 4-8 所示。

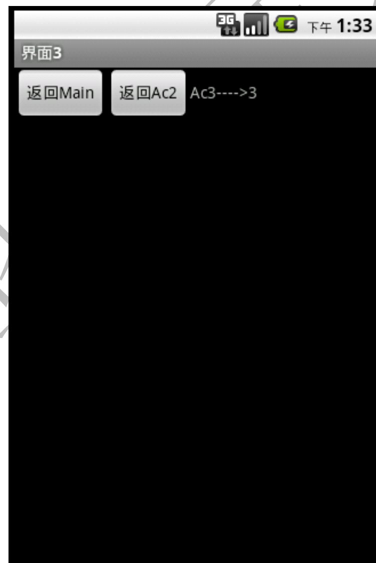


图 4-8　Activity3 的界面

如图 4-8 所示，单击“返回 Main”则跳转至 Fx_Main，单击“返回 AC2”，则跳转到 Activity2。具体代码如代码清单 4-5 所示。

代码清单 4-5　Activity3.Activity

```
import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.TextView;
```

```
public class Activity3 extends Activity {
  private Button b;
  private Button b2;
  @Override
protected void onCreate(Bundle savedInstanceState) {
  // TODO Auto-generated method stub
  super.onCreate(savedInstanceState);
  setContentView(R.layout.activity3);
  b=(Button)findViewById(R.id.Button03);
  b2=(Button)findViewById(R.id.Button04);
  TextView tv=(TextView)findViewById(R.id.TextView03);
  tv.setText("Ac3---->"+getTaskId());
  Log.i("System.out", "Ac3---->"+this.toString()+"Task ID---->"+getTaskId());
  b.setOnClickListener(new OnClickListener(){

       @Override
       public void onClick(View v) {
            // TODO Auto-generated method stub
            Intent i=new Intent(Activity3.this,Fx_Main.class);
            startActivity(i);
       }});
  b2.setOnClickListener(new OnClickListener(){

       @Override
       public void onClick(View v) {
            // TODO Auto-generated method stub
            Intent i=new Intent(Activity3.this,Activity2.class);
            startActivity(i);
       }});
}
  @Override
  protected void onDestroy() {
       // TODO Auto-generated method stub
       super.onDestroy();
  Log.i("System.out", "Ac3--->Destory");
  }
}
```

4.4.1 standard 标准模式

在 standard 模式也就是默认模式下，不需要配置 launchMode。此时的 AndroidManifest.xml 如代码清单 4-6 所示。

代码清单 4-6 AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
     package="feixun.com.jiang"
     android:versionCode="1"
     android:versionName="1.0">
   <application android:icon="@drawable/icon" android:label="@string/app_name">
      <activity android:name=".Fx_Main"
              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" android:label="@string/Ac2"/ >
      <activity android:name=".Activity3" android:label="@string/Ac3/>

   </application>
   <uses-sdk android:minSdkVersion="4" />

</manifest>
```

运行例子，从 Fx_Main 开始，一直点回到 Activity2 按钮时，Log 信息如图 4-9 所示。

Time		pid	tag	Message
05-02 ...	I	310	Syste...	Main---->Fx_Main@43e09d38Task ID---->4
05-02 ...	I	310	Syste...	Ac2---->Activity2@43e16dd0Task ID---->4
05-02 ...	I	310	Syste...	Ac2---->Activity2@43e1f318Task ID---->4
05-02 ...	I	310	Syste...	Ac2---->Activity2@43e26788Task ID---->4
05-02 ...	I	310	Syste...	Ac2---->Activity2@43e2dc10Task ID---->4

图 4-9　Standard 启动模式下 Log 信息

发现每次都创建了 Activity2 的新实例。standard 的加载模式就是这样的，Intent 将发送给它新的 Activity 实例。

现在点击 Android 设备的回退键，可以看到 Log 信息按照刚才创建 Activity 实例的倒序依次出现，类似退栈的操作，而刚才操作跳转按钮的过程是压栈的操作。

4.4.2　singleTop

singleTop 和 standard 模式，都会将 Intent 发送到新的实例（如果已经有了，singleTask 模式和 singleInstance 模式不发送到新的实例）。不过，singleTop 要求如果创建 intent 时栈顶已经有要创建 Activity 的实例，则将 Intent 发送给该实例，而不发送给新的实例。

还是用刚才的示例，只需将 Activity2 的 launchMode 改为 singleTop，就能看到区别。修改后 AndroidManifest.xml 中代码如代码清单 4-7 所示。

代码清单 4-7　AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
     package="feixun.com.jiang"
     android:versionCode="1"
     android:versionName="1.0">
   <application android:icon="@drawable/icon" android:label="@string/app_name">
      <activity android:name=".Fx_Main"
              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" android:label="@string/Ac2"
                   android:launchMode="singleTop"/ >
      <activity android:name=".Activity3" android:label="@string/Ac3/>

   </application>
   <uses-sdk android:minSdkVersion="4" />

</manifest>
```

运行 Fx_Main，跳转到 Activity2---->Actvity2 时会发现，单击多少遍按钮，都是相同的 Activity2 实例，因为该实例在栈顶，所以不会创建新的实例。如果回退，回到 Fx_Main，将退出应用，如图 4-10 所示。

```
05-02 ...   I   339   Syste...   Main---->  Fx_Main@43e09d38Task ID---->5
05-02 ...   I   339   Syste...   Ac2----> Activity2@43e16db8Task ID---->5
```

图 4-10　singleTop 模式下“跳转到 AC2”的 Log 信息

singleTop 模式，可用来解决栈顶多个重复相同的 Activity 的问题。

如果是 Fx_Main 跳转到 Activity2，再跳转到 Fx_Main，行为就和 standard 一样了，会在 Activity2 跳转到 Fx_Main 时创建 Fx_Main 的新实例，因为当时的栈顶不是 Activity2 实例，如图 4-11 所示。

```
05-02 ...   I   339   Syste...   Main---->  Fx_Main@43e27960Task ID---->6
05-02 ...   I   339   Syste...   Ac2----> Activity2@43e2d590Task ID---->6
05-02 ...   I   339   Syste...   Main---->  Fx_Main@43e34960Task ID---->6
```

图 4-11　singleTop 模式下“跳转到 AC2”后“跳回到 Main”的 Log 信息

4.4.3 singleTask

singleTask 模式和后面的 singleInstance 模式都是只创建一个实例的。

当 Intent 到来，需要创建 singleTask 模式 Activity 时，系统会检查栈里面是否已经有该 Activity 的实例。如果有直接将 Intent 发送给它（注意此时原在此 Activity 栈中上面的 Activity 将会被关闭）。

把 Activity2 的启动模式改成 singleTask，修改后 AndroidManifest.xml 中代码如代码清单 4-8 所示。

代码清单 4-8　AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
     package="feixun.com.jiang"
     android:versionCode="1"
     android:versionName="1.0">
   <application android:icon="@drawable/icon" android:label="@string/app_name">
      <activity android:name=".Fx_Main"
              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" android:label="@string/Ac2"
                   android:launchMode="singleTask"/ >
     <activity android:name=".Activity3" android:label="@string/Ac3/>

   </application>
   <uses-sdk android:minSdkVersion="4" />

</manifest>
```

启动 Fx_Main，跳转到 Activity2---->Activity3---->Actvity2，此时看 Log 信息，如图 4-12 所示。

```
05-02 ...  I  398  Syste...  Main----> Fx_Main@43e09d38Task ID---->8
05-02 ...  I  398  Syste...  Ac2----> Activity2@43e16dd0Task ID---->8
05-02 ...  I  398  Syste...  Ac3----> Activity3@43e1f550Task ID---->8
05-02 ...  I  398  Syste...  Activity3---->onDestroy
```

图 4-12　singleTask 启动模式下 Log 信息

可见从 AC3 再跳转到 AC2 时，因为 AC2 之前在栈中是存在的所以不生成新的 AC2 实例，而是在栈中找到此 AC2，并将在 AC2 上面的 AC3 关闭，所以此时栈中只有 Fx_Main 和 AC2，在 AC2 点返回会直接退到 Fx_Main 然后退出。

4.4.4 singleInstance

在 singleInstance 模式下，加载该 Activity 时如果没有实例化，它会在创建新的 Task 后，实例化入栈，如果已经存在，则直接调用 onNewIntent，该 Activity 的 Task 中不允许启动其他的 Activity，任何从该 Activity 启动的其他 Activity 都将被放到其他 Task 中，先检查是否有在应用的 Task，没有的话就创建。

在这里介绍一下 Task（任务）的概念。按照字面意思，任务就是自己要实现的一个目的，而在 Android 中的 Task 的定义是一系列 Activity 的集合，即要达到自己最终要到的 Actvity，之前所有经历过的 Actvity 的集合。它可以是同一个应用内部的，也可以是两个不同应用的。Task 可以认为是一个栈，可放入多个 Activity。比如，启动一个应用，那么 Android 就创建了一个 Task，然后启动这个应用的入口 Activity，就是 intent-filter 中配置为 main 和 launch 的那个。这个 Activity 是根（Root）Activity，可能会在它的界面调用其他 Activity，这些 Activity 如果按照上面那 3 个模式，也会在这个栈（Task）中，只是实例化的策略不同而已。

把 Activity2 的启动模式改成 singleInstance，修改后 AndroidManifest.xml 中代码如代码清单 4-9 所示。

代码清单 4-9　AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
```

```
        package="feixun.com.jiang"
        android:versionCode="1"
        android:versionName="1.0">
      <application android:icon="@drawable/icon" android:label="@string/app_name">
          <activity android:name=".Fx_Main"
                  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" android:label="@string/Ac2"
                      android:launchMode="singleInstance"/ >
         <activity android:name=".Activity3" android:label="@string/Ac3/>

      </application>
      <uses-sdk android:minSdkVersion="4" />

  </manifest>
```

然后进行测试，启动 Fx_Main---->Actvity2---->Actvity3 然后看一下 Log 信息，如图 4-13 所示。

```
05-02 ...  I  426  Syste...  Main---->.Fx_Main@43e09d38Task ID---->9
05-02 ...  I  426  Syste...  Ac2---->  Activity2@43e16dd0Task ID---->10
05-02 ...  I  426  Syste...  Ac3---->  Activity3@43e1f000Task ID---->9
```

图 4-13 singleInstance 启动模式下 Log 信息

可以看到 Fx_Main 以及 Activity3 的 Task ID 为 9，而 Actvity2 的 Task ID 为 10，此时在 Actvity3 单击“返回”按钮会发现先退到 Fx_Main，继续返回会回到 Actvity2 最后退出。从该过程可以看出：如果从其他应用程序调用 singleInstance 模式的 Activity（Fx_Main），从该 Activity 开启其他 Activity（Activity2）时，会创建一个新的 Task（Task ID 为 10 的那个），实际上，如果包含该 Activity（Activity2）的 Task 已经运行的话，他会在该运行的 Task 中重新创建。

经过上述的介绍，用下面的表格来进行一个简单的总结，如表 4-2 所示。

表 4-2 Activity4 种启动模式对比

区别	是否允许多个实例	如何决定所属 Task	是否每次都生成新实例	是否允许其他 Activity 存在于本 Task 内
standard	可被多次实例化，同一个 Task 的不同的实例可位于不同的 Task 中，每个 Task 也可包含多个实例	存放于 Start Activity() 的 Task。除非设置 FLAG_ ACTIVITY_NEW_TASK 标记	是	允许
singleTop	同 standard	同 standard	如果寄存 Activity 的栈顶为该 Activity，则直接用该 Activity 处理；否则，创建新实例	允许
singleTask	不能有多个实例。由于该模式下 Activity 总是位于栈顶，所以 Actvity 在同一个设备里最多只有一个实例	放入新的 Task 内，并且位于该 Task 的根	只有在第一次才创建新的实例，其他情况复用该 Activity	允许。如果存放 singleTask 的栈寄存在 Task 内，响应一个 Intent 时，如果 singleTask 位于栈顶，则处理 Intent，否则会丢失 Intent，但该 Task 会处于前台
singleInstance	同 singleTask	同 singleTask	同 singleTask	不允许

4.5 程序调试

Android 系统提供了两种调试工具 LogCat 和 DevTools，用于定位、分析及修复程序中出现的错误。

4.5.1 LogCat 命令行工具

LogCat 是可以显示在 Eclipse 集成开发环境中的用来获取系统日志信息的工具。它的主要功能就是能够捕获包括 Dalvik 虚拟机产生的信息、进程信息、ActivityManager 信息、PackagerManager 信息、Homeloader 信息、WindowsManager 信息、Android 运行时信息和应用程序信息等可被捕获的信息。

1. LogCat 的使用方法

打开方式：选择“Window”→“Show View ”→“Other”命令，打开 Show View 的选择菜单，然后在 Andoird → LogCat 中选择 LogCat。打开 LogCat 后，它便显示在 Eclipse 的下方区域，其界面如图 4-14 所示。

图 4-14 LogCat 界面

从图中我们可以看到 LogCat 的右上方有 5 个不同的字母，这 5 个字母分别表示 5 种不同类型的日志信息，它们的级别依次增高，表示含义如下。

- V：详细（Verbose）信息。
- D：调试（Debug）信息。
- I：通告（Info）信息。
- W：警告（Warn）信息。
- E：错误（Error）信息。

在 LogCat 中，用户可以通过 5 个字母图标选择显示的信息类型，级别高于所选类型的信息也会在 LogCat 中显示，但级别低于所选类型的信息则不会被显示。

同时，LogCat 提供了“过滤”功能，在右上角的“+”号和“-”号，分别是添加和删除过滤器。用户可以根据日志信息的标签（Tag）、产生日志的进程编号（Pid）或信息等级（Level），对显示的日志内容进行过滤。

2. 程序调试原理

- 引入 android.util.Log 包。
- 使用 Log.v()、 Log.d()、 Log.i() 、Log.w() 和 Log.e() 5 个方法在程序中设置“日志点”。
  - Log.v()用来记录详细信息。
  - Log.d()用来记录调试信息。
  - Log.i()用来记录通告信息。
  - Log.w()用来记录警告信息。
  - Log.e()用来记录错误信息。
- 当程序运行到“日志点”时，应用程序的日志信息便被发送到 LogCat 中。
- 判断“日志点”信息与预期的内容是否一致。
- 进而判断程序是否存在错误。

下面的例子演示了 Log 类的具体使用方法。

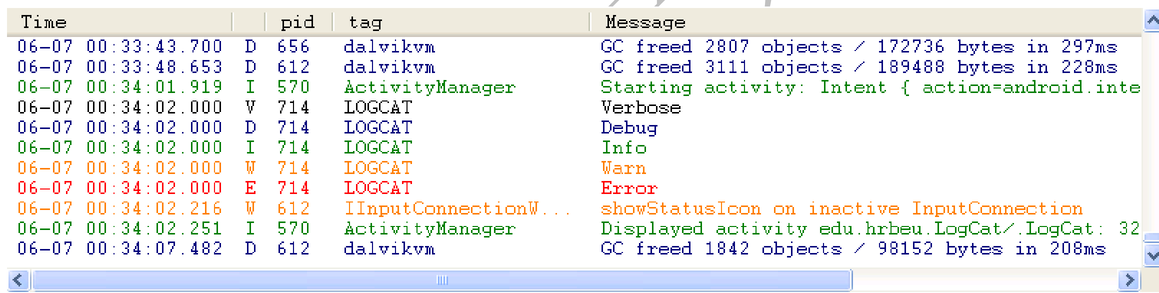
代码清单 4-10 LogCat.java

华清远见

```
package com.example.LogCat;

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

public class LogCat extends Activity {
    final static String TAG = "LOGCAT";
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        Log.v(TAG,"Verbose");
        Log.d(TAG,"Debug");
        Log.i(TAG,"Info");
        Log.w(TAG,"Warn");
        Log.e(TAG,"Error");
    }
}
```

在本段代码中，程序第 5 行“import android.util.Log;”引入 android.util.Log 包；第 8 行定义标签，标签帮助用户在 LogCat 中找到目标程序生成的日志信息，同时也能够利用标签对日志进行过滤；第 14 行记录一个详细信息，Log.v()方法的第一个参数是日志的标签，第二个参数是实际的信息内容；第 15~18 行分别产生了调试信息、通告信息、警告信息和错误信息。

最终运行结果如图 4-15 所示，从图中还可以看出 LogCat 对不同类型的信息使用了不同的颜色加以区别。



Time		pid	tag	Message
06-07 00:33:43.700	D	656	dalvikvm	GC freed 2807 objects / 172736 bytes in 297ms
06-07 00:33:48.653	D	612	dalvikvm	GC freed 3111 objects / 189488 bytes in 228ms
06-07 00:34:01.919	I	570	ActivityManager	Starting activity: Intent { action=android.inte
06-07 00:34:02.000	V	714	LOGCAT	Verbose
06-07 00:34:02.000	D	714	LOGCAT	Debug
06-07 00:34:02.000	I	714	LOGCAT	Info
06-07 00:34:02.000	W	714	LOGCAT	Warn
06-07 00:34:02.000	E	714	LOGCAT	Error
06-07 00:34:02.216	W	612	IInputConnectionW...	showStatusIcon on inactive InputConnection
06-07 00:34:02.251	I	570	ActivityManager	Displayed activity edu.hrbeu.LogCat/.LogCat: 32
06-07 00:34:07.482	D	612	dalvikvm	GC freed 1842 objects / 98152 bytes in 208ms

图 4-15 LogCat 工程的运行结果

3. 添加过滤器

上文中提到 LogCat 提供了“过滤”功能，下面就来介绍一下 LogCat 是如何添加过滤器的。

首先，单击右上角的“+”，在弹出的对话框中填入过滤器的名称：LogcatFilter，设置过滤条件为“标签=LOGCAT”即可，操作方法如图 4-16 所示。

图 4-16 添加过滤器

经过上述过滤器过滤后，无论什么类型的日志信息，属于哪一个进程，只要标签为 LogCat，都将显示在 LogcatFilter 区域内。LogCat 过滤后的输入结果如图 4-17 所示。

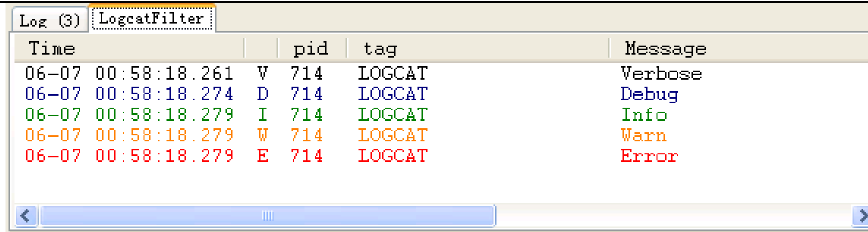


图 4-17 LogCat 过滤后的输入结果

4.5.2 DevTools 开发调试工具

DevTools 是用于调试和测试的工具，它包括了如下所示一系列各种用途的用户小工具：Development Settings、Exception Browser、Google Login Service、Instrumentation、Media Scanner、Package Browser、Pointer Location、Raw Image Viewer、Running processes 和 Terminal Emulator。

如图 4-18 所示，为 DevTools 使用时的界面。由使用时的界面也可以看出其中的各个小工具。

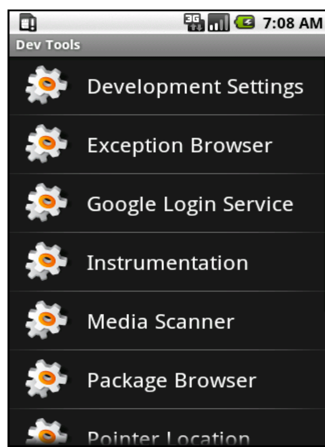


图 4-18 DevTools 的使用界面

以下着重讲解 Dev Tools 的一些小工具。

1．Development Settings

Development Settings 中包含了程序调试的相关选项，单击功能前面的选择框，出现绿色的“对号”表示功能启用，模拟器会自动保存设置。

图 4-19 显示了 Development Settings 的运行界面。

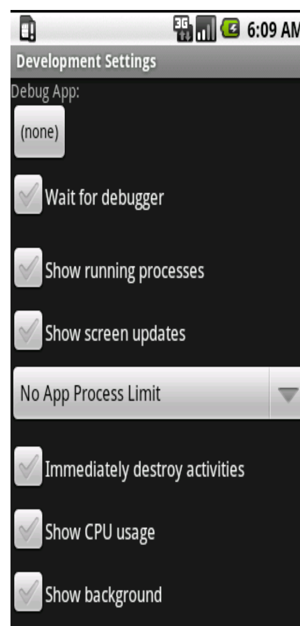


图 4-19　Development Settings 运行界面

下面就详细介绍 Development Settings 中各个选项的含义，如表 4-3 所示。

表 4-3　Development Settings 中各选项的含义

选项	说明
Debug App	为 Wait for debugger 选项指定应用程序，如果不指定（选择 none），Wait for debugger 选项将适用于所有应用程序。Debug App 可以有效地防止 Android 程序长时间停留在断点而产生异常
Wait for debugger	阻塞加载应用程序，直到关联到调试器（Debugger）。用于在 Activity 的 onCreate()方法的进行断点调试
Show running processes	在屏幕右上角显示运行中的进程
Show screen updates	选中该选项时，界面上任何被重绘的矩形区域会闪现粉红色，有利于发现界面中不必要的重绘区域
No App Process limit	允许同时运行进程的数量上限
Immediately destroy activities	Activity 进入停止状态后立即销毁，用于测试在方法 onSaveInstanceState()、onRestoreInstanceState()和 onCreate()中的代码
Show CPU usage	在屏幕顶端显示 CPU 使用率，上层红线显示总的 CPU 使用率，下层绿线显示当前进程的 CPU 使用率
Show background	应用程序没有 Activity 显示时，直接显示背景面板，一般这种情况仅在调试时出现
Show Sleep state on LED	在休眠状态下开启 LED
Windows Animation Scale	窗口动画规模
Transition Animation	转换动画
Light Hinting	轻显示
Show GTalk service connection status	显示 GTalk 服务连接状态

2. Pointer Location

Pointer Location 是屏幕点位置查看工具，能够显示触摸点的 X 轴坐标和 Y 轴坐标，如图 4-20 所示。

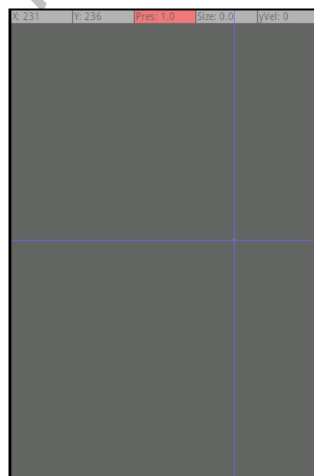


图 4-20　Pointer Location 的使用画面

3. Running processes

Running processes 能够查看在 Android 系统中正在运行的进程，并能查看进程的详细信息，包括进程名称和进程所调用的程序包。

华清远见

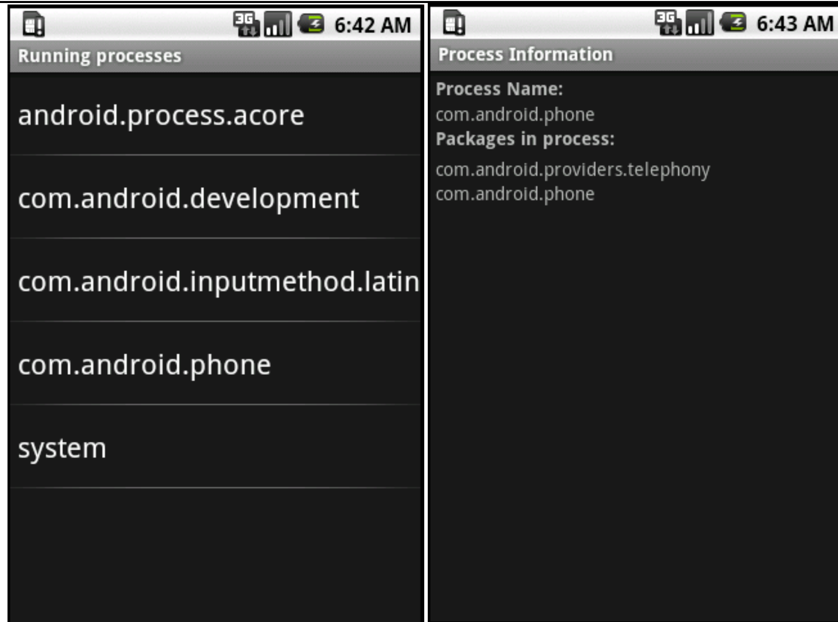


图 4-21 Andoird 模拟器默认情况下运行的进程和 com.android.phone 进程的详细信息

4．Terminal Emulator

Terminal Emulator 可以打开一个连接底层 Linux 系统的虚拟终端，但具有的权限较低，且不支持提升权限的 su 命令。如果需要使用 root 权限的命令，可以使用 ADB 工具。

图 4-22 是 Terminal Emulator 运行时的画面，输入 ls 命令，显示出根目录下的所有文件夹。

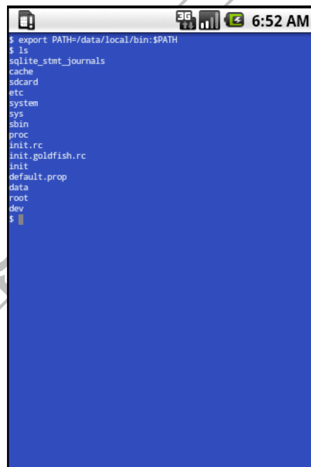


图 4-22 Terminal Emulator 运行时的画面

4.6 本章小结

本章主要介绍了 Android 系统的进程优先级排序、不同优先级进程之间的变化方式，Android 系统的 4 大基本组件及其用途，Activity 的生命周期中各个状态及状态间的变化关系、Android 应用程序的调试方法和工具。

关键知识点测评

1．以下有关 Android 系统进程优先级的说法，不正确的一个是（ ）。

A．前台进程是 Android 系统中最重要的进程

B．空进程在系统资源紧张时会被首先清除

C．服务进程没有用户界面并且在后台长期运行

D．Android 系统中一般存在数量较多的可见进程

2．以下有关 Android 组件的叙述，正确的一个是（　　）。

A．Service 是 Android 程序的呈现层

B．BroadcaseReceiver 本身包含界面，用于通知用户接收到重要信息

C．应用程序可以通过 ContentProvider 访问其他应用程序的私有数据

D．不是所有的 Android 组件都具有自己的生命周期

3．以下有关 Activity 生命周期的描述，不正确的是（　　）。

A．Activity 的状态之间是可以相互转换的

B．Activity 的全生命周期是从 Activity 建立到销毁的全部过程，始于 onCreate()，结束于 onDestroy()

C．活动生命周期是 Activity 在屏幕的最上层，并能够与用户交互的阶段

D．onPause()函数在 Android 系统因资源不足终止 Activity 前调用

联系方式

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

华清远见