第5章 实现多任务

5.1 基本概念

在安卓系统中，如果有一个应用程序组件是第一次被启动，而且这时候，应用程序也没有其他组件在运行，则安卓系统会为应用程序创建一个 Linux 进程，这个 Linux 进程只包含一个线程。举个例子，如果一个应用程序启动了第一个活动，这个活动里有一个文本框和一个按钮，这时安卓系统会为应用程序创建一个单线程的 Linux 进程，初始化这个文本框和按钮，当这个应用程序启动另一个活动时，初始化图形组件的还是这个已经创建好的线程，不会再创建新的。也就是说，这个应用程序会一直单线程单任务运行图形组件的初始化和与图形组件相关的操作。

默认情况下，同一个应用程序的所有组件都运行在同一个进程和线程里，这个线程叫作主线程。如果一个组件启动时，应用程序的其他组件已经在运行了，则此组件会在已有的进程和线程中启动运行。如果希望安卓应用程序实现多任务，可以通过代码指定组件运行在其他进程里，或为进程创建额外的线程。下面介绍安卓的进程调度机制。

5.1.1 进程

默认情况下，同一个应用程序内的所有组件都是运行在同一个进程中的，大部分应用程序都是按照这种方式运行。但是在具体应用中，很多时候需要通过在 AndroidManifest.xml 文件中进行设置，指定某个特定组件归属于哪个进程。可以通过 AndroidManifest.xml 文件设定应用程序归属的进程。AndroidManifest.xml 文件中的每一种组件元素——<activity>、<service>、<receiver>和<provider>——都支持定义 android:process 属性，用于指定组件运行的进程。设置这个属性就可实现每个组件在各自的进程中运行，或者某几个组件共享一个进程而其他组件运行于独立的进程。设置这个属性也可以让不同应用程序的组件运行在同一个进程中，这就实现了多个应用程序共享同一个 Linux 用户 ID、赋予同样的权限。<application>元素也支持 android:process 属性，用于指定所有组件的默认进程。

安卓一个重要并且特殊的特性就是，一个应用的进程的生命周期不是由应用程序自身直接控制的，而是由系统根据运行中的应用的一些特征来决定的，包括这些应用程序对用户的重要性、系统的全部可用内存。大部分情况下，每个安卓应用程序都将运行在自己的 Linux 进程当中。当这个应用的某些代码需要执行时，进程就会被创建，并且将保持运行，直到该进程不再需要，而系统需要释放它所占用的内存，为其他应用所用时才停止。

安卓系统试图尽可能长时间地保持应用程序进程，但为了新建或者运行更加重要的进

程，总是需要清除过时进程来回收内存。为了决定保留或终止哪个进程，根据进程内运行的组件及这些组件的状态，系统把每个进程都划入一个重要性层次结构中。重要性最低的进程首先会被清除，然后是下一个最低的，以此类推，这都是回收系统资源所必需的。重要性层次结构共有五级，以下按照重要程度列出了各类进程，其中第一类进程是最重要的：前台进程＞可见进程＞服务进程＞后台进程＞空进程。

1. 前台进程

用户当前操作所必需的进程。满足以下任一条件时，进程被视作处于前台。

（1）正在与用户交互的活动进程（例如：活动的 onResume()方法已被调用）。

（2）正在与用户交互的活动绑定的服务进程。

（3）正在运行前台 Service 进程，例如服务被 startForeground()方法调用。

（4）正在运行生命周期回调方法的服务，例如 onCreate()、onStart()或 onDestroy()。

（5）正在运行 onReceive()方法的广播接收器。

一般而言，任何时刻只有很少的前台进程同时运行。只有当内存不足以维持它们同时运行时，作为最后的策略它们才会被终止。通常，终止一些前台进程是为了保证用户界面的及时响应。

2. 可见进程

如果没有任何前台组件但仍会影响用户在屏幕上所见内容的进程，称为可见进程。满足以下任一条件时，进程被认为是可见的：如果活动不在前台但用户仍然可见。例如，当前台活动打开了一个对话框，而之前的活动还允许显示在后面，但是已经无法与用户进行交互了。例如，活动的 onPause()方法被调用了；一个绑定到可见或前台活动的服务进程。可见进程被认为是非常重要的进程，除非无法维持所有前台进程同时运行了，它们是不会被终止的。

3. 服务进程

对于由 startService()方法启动的服务进程不会升级为上述两种级别。尽管服务进程不直接和用户所见内容关联，但它们通常在执行一些用户关心的操作。例如，在后台播放音乐或从网络下载数据等，因此除非内存不足以维持所有前台、可见进程同时运行，系统会保持服务进程的运行。

4. 后台进程

包含目前用户不可见活动的进程。例如，活动的 onStop()方法已被调用。这些进程对用户体验没有直接的影响，系统可能在任意时间终止它们，以回收内存供前台进程、可见进程及服务进程使用。通常会有很多后台进程在运行，所以被保存在一个最近最少使用列表中，以确保最近被用户使用的活动最后一个被终止。如果一个活动正确实现了生命周期方法，并保存了当前的状态，则终止此类进程不会对用户体验产生可见的影响，因为在用户返回时活动会恢复所有可见的状态。

5. 空进程

不含任何活动应用程序组件的进程，保留这种进程的唯一目的就是用作缓存，以改善下次在此进程中运行组件的启动时间。为了在进程缓存和内核缓存间平衡系统整体资源，系统经常会终止这种进程。依据进程中目前活跃组件的重要程度，安卓会给进程评估一个尽可能高的级别。例如，如果一个进程中运行着一个服务和一个用户可见的活动，则此进程会

被评定为可见进程而不是服务进程。此外，一个进程的级别可能会由于其他进程的依赖而被提高，为其他进程提供服务的进程级别永远不会低于使用此服务的进程。因为运行服务的进程级别是高于后台活动进程的，所以如果活动需要启动一个长时间运行的操作，则为其启动一个服务会比简单地创建一个工作线程更好些，尤其是在此操作时间比活动本身存在时间还要长久的情况下。

5.1.2　线程

应用程序启动时，系统会为其创建一个名为“main”的主线程。主线程非常重要，因为其负责把事件分发给相应的用户界面（包括屏幕绘图事件），也是应用程序与安卓界面组件包（来自 android.widget 和 android.view 包）进行交互的线程，因此主线程有时也被叫作界面线程。系统并不会为每个组件的实例都创建单独的线程。运行于同一个进程中的所有组件都是在界面线程中实例化的，对每个组件的系统调用也都是由界面线程分发的。

如果应用程序在与用户交互的同时需要执行繁重的任务，用户单线程模式可能会导致运行性能很低下。例如，在查询数据库时，应用程序就需要做两件事，一是需要与数据库连接，访问数据库，获取查询结果；二是要初始化显示界面的组件，把获取的数据给显示出来。因为是单线程，就必须先做完第一件事后才能做第二件事。这个过程有可能因为网络状况或数据库繁忙，在访问数据库、获取结果数据时花费比较长的时间，导致不能执行用户显示界面的初始化，使得用户界面呈现出静止状态。这种状态称为界面线程阻塞。如果界面线程被阻塞超过一定时间（目前大约是 5s），用户就会被提示“应用程序没有响应”（ANR），如图 5-1 所示。

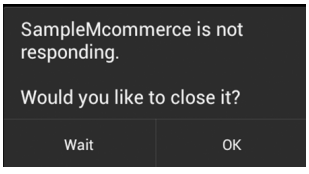

图 5-1　界面线程阻塞

安卓的单线程模式遵守以下两个规则。

（1）不要阻塞界面线程。

（2）不要在界面线程之外访问安卓的界面组件包。

这样程序才能有友好的界面顺利运行。一般稍微复杂一点的应用程序，特别是需要网络访问或数据库访问的应用程序，都需要使用多任务的方式。在安卓应用程序中，创建的活动、服务、广播接收器等都是在主线程（界面线程）处理的，但一些比较耗时的操作，如大文件读写、数据库操作以及网络下载都需要很长时间，为了不阻塞用户界面，出现 ANR 的响应提示窗口，这个时候可以考虑创建一个工作线程来解决，继承 Thread 类或者实现 Runnable 接口。

5.2　实现多任务

安卓多任务的调度和实现采用消息驱动机制。熟悉 Windows 编程的读者可能知道 Windows 程序是消息驱动的，并且有全局的消息循环系统。而安卓应用程序也是消息驱动的，谷歌参考了 Windows 系统，也在安卓系统中实现了消息循环机制。安卓通过 Looper、Handler、MessageQueue 和 Message 来实现消息循环机制，安卓消息循环是针对线程的，就是说，主线程和工作线程都可以有自己的消息队列和消息循环。

5.2.1 实现原理

对于多线程的安卓应用程序来说有两类线程：一类是主线程，也就是界面线程；另一类是工作线程，也就是主线程或工作线程所创建的线程。安卓的线程间消息处理机制主要是用来处理主线程跟工作线程间通信的，图 5-2 是线程间通信原理图。安卓应用程序是通过消息来驱动的，即在应用程序的主线程中有一个消息循环，负责处理消息队列中的消息，例如，当从网上下载文件时，为了不使主线程被阻塞，通常需要创建一个子线程来负责下载任务，同时在下载的过程中将下载进度以百分比的形式在应用程序的界面上显示出来，这样既不会阻塞主线程的运行，又能获得良好的用户体验，但是安卓应用程序的子线程是不可以操作主线程的界面的，那么这个负责下载任务的子线程应该如何在应用程序界面上显示下载的进度呢？如果能够在子线程中往主线程的消息队列中发送消息，那么问题就迎刃而解了，因为发往主线程消息队列的消息最终是由主线程来处理的，在处理这个消息时，就可以在应用程序界面上显示下载进度了。

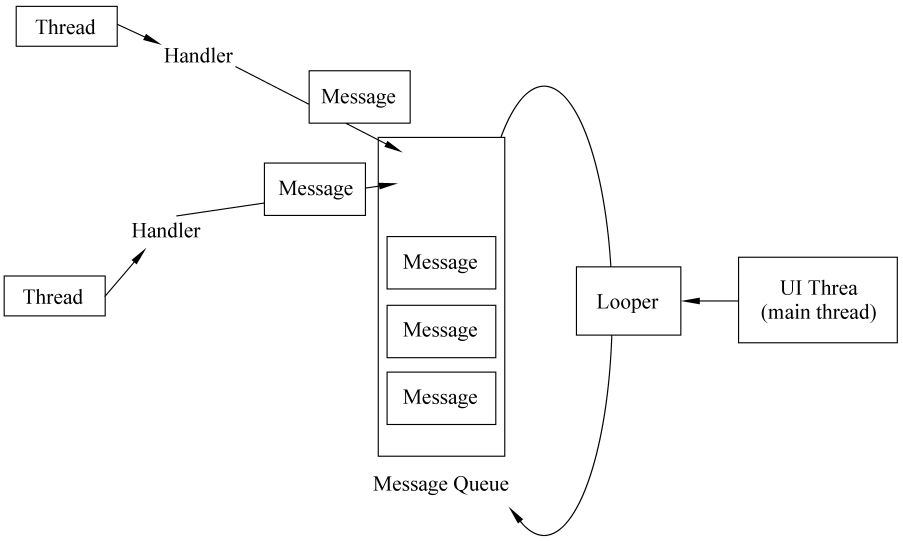

图 5-2 线程间通信原理

线程之间和进程之间是不能直接传递消息的，必须通过对消息队列和消息循环的操作来完成。安卓消息循环是针对线程的，每个线程都可以有自己的消息队列和消息循环。安卓提供了 Handler 类和 Looper 类来访问消息队列 Message Queue。Looper 类是用来封装消息循环和消息队列的一个类，负责管理线程的消息队列和消息循环，用于在安卓线程中进行消息处理。Looper 对象是什么呢？其实安卓中每一个线程都对应一个 Looper，Looper 可以帮助线程维护一个消息队列，是负责在多线程之间传递消息的一个循环器，线程通过 Looper 对象可以读写某个消息循环队列。使用 Looper.myLooper()得到当前线程的 Looper 对象，使用 Looper.getMainLooper()可以获得当前进程的主线程的 Looper 对象。

一个线程可以存在，也可以不存在一个消息队列和一个消息循环，工作线程默认是没有消息循环和消息队列的，如果想让工作线程具有消息队列和消息循环，需要在线程中首先调用 Looper.prepare()来创建消息队列，然后调用 Looper.loop()进入消息循环，见代码 5-1。

```
import android.os.Bundle;
import android.os.Handler;
import android.os.Looper;
import android.os.Message;
import android.util.Log;
import android.view.View;
import android.view.View.OnClickListener;
import androidx.appcompat.app.AppCompatActivity;
import com.example.ch06.R;
public class LooperThreadActivity extends AppCompatActivity {
    private final int MSG_HELLO = 0;
    private Handler mHandler;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.c06_handler_test03);
        new CustomThread().start();//新建并启动 CustomThread 实例
        findViewById(R.id.send_btn).setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {//单击界面时发送消息
                String str = "hello";
                Log.d("Test", "MainThread is ready to send msg:" + str);
                //发送消息到 CustomThread 实例
                mHandler.obtainMessage(MSG_HELLO, str).sendToTarget();
            }
        });
    }
    class CustomThread extends Thread {
        @Override
        public void run() {
            //建立消息循环的步骤
            //1. 初始化 Looper
            Looper.prepare();
            //2. 绑定 handler 到 CustomThread 实例的 Looper 对象
            mHandler = new Handler() {
                //3. 定义处理消息的方法
                public void handleMessage(Message msg) {
                    switch (msg.what) {
                        case MSG_HELLO:
                        Log.d("Test", "CustomThread receive msg:"
                                    + (String) msg.obj);
                    }
                }
            };
            //4. 启动消息循环
            Looper.loop();
        }
    }
}
```

代码 5-1　CustomThread.java

通过代码 5-1 的设置，工作线程 CustomThread 就具有了消息队列和消息循环的处理

机制了，可以在 Handler 中进行消息处理。代码中定义的 Handler 对象，其作用是把消息加入特定的消息队列中，并分发和处理该消息队列中的消息。

```
2022-03-24 11:22:21.536 7370-7370/com.example.ch06 D/Test: MainThread is ready
to send msg:hello
2022-03-24 11:22:21.536 7370-7495/com.example.ch06 D/Test: CustomThread
receive msg:hello
```

代码 5-2　日志输出

每个活动是一个界面线程，运行于主线程中。安卓系统在启动的时候会为活动创建一个消息队列和消息循环。一个活动中可以创建多个工作线程或者其他的组件，如果这些线程或者组件把它们的消息放入活动的主线程消息队列，那么该消息就会在主线程中处理了。因为主线程一般负责界面的更新操作，并且安卓系统中的界面控件都是单线程模式，多线程控制需要程序员实现，也就是非线程安全的，所以这种方式可以很好地实现安卓界面更新。在安卓系统中这种机制有着广泛的运用，一个工作线程是通过 Handle 对象把消息放入主线程的消息队列。只要 Handler 对象由主线程的 Looper 创建，那么调用 Handler 的 sendMessage()等方法，就会把消息放入主线程的消息队列。在主线程中调用 Handle 的 handleMessage()方法来处理消息，在这个方法中实现主线程的界面控件的操作，从而实现了工作线程和主线程之间的调度。在活动 LooperThreadActivity 中定义了 Handler 对象 mHandler，并定义了一个工作进程 CustomThread，在工作进程中使用对象 mHandler 的 sendMessage()发送了一条消息到主线程的消息队列。

```
2022-03-24 11:22:21.536 7370-7370/com.example.ch06 D/Test: MainThread is ready
to send msg:hello
2022-03-24 11:22:21.536 7370-7495/com.example.ch06 D/Test: CustomThread
receive msg:hello
```

代码 5-3　日志调试输出

可以看到有两条调试信息，显示了在这个活动运行过程中，各个模块所处的线程情况。在这个例子中，主线程在 onCreate()方法中通过“new CustomThread().start()”启动了工作线程，工作线程 CustomThread 中 run()方法执行代码，访问了主线程 Handler 对象 mHandler，并在调用 Handler 的对象 mHandler 时，向主线程消息队列加入了一条消息。因为 Handler 对象管理的 Looper 对象是线程安全的，不管是加入消息到消息队列或者是从队列读出消息都是有同步对象保护的，由于这里没有修改 Handler 对象，所以 Handler 对象不可能会出现数据不一致的问题。

工作线程和主线程运行在不同的线程中，所以必须要注意这两个线程间的竞争关系。在主线程中构造 Handler 对象，并且启动工作线程之后不要再修改，否则会出现数据不一致。这样在工作线程中可以放心地调用发送消息 sendMessage()方法传递消息，Handler 对象的 handleMessage()方法将会在主线程中调用。在这个方法中可以安全地调用主线程中任何变量和函数，进而完成更新界面的任务。安卓有以下两种方式实现多线程操作界面。

(1) 第一种是创建新线程 Thread，用 Handler 负责线程间的通信和消息。

(2) 第二种方式是 AsyncTask 异步执行任务。

5.2.2　Handler

首先来看看如何使用 Handle 实现多任务。android.os.Handler 是安卓中处理定时操作的核心类。通过 Handler 类，可以提交和处理一个 Runnable 对象。这个对象的 run()方法可以立刻执行，也可以在指定时间之后执行(可以称为预约执行)。Handler 类有以下两种主要用途。

(1) 按照时间计划，在未来某时刻，处理一个消息或执行某个 Runnable 实例。

(2) 把一个对另外线程对象的操作请求放入消息队列中，从而避免线程间冲突。

当一个进程启动时，主线程独立执行一个消息队列，该队列管理着应用顶层的对象(例如活动、广播接收器等)和所有创建的窗口。可以创建自己的一个线程，并通过 Handler 来与主线程进行通信。这可以通过在新的线程中调用主线程的 Handler 的 postXXX()和 sendMessage()方法来实现，使用 post()方法实现多任务的主要步骤如下(如代码 5-4 所示)。

(1) 创建一个 Handler 对象。

(2) 将要执行的操作写在线程对象的 run()方法中。

(3) 使用 post()方法运行线程对象。

(4) 如果需要循环执行，需要在线程对象的 run()方法中再次调用 post()方法。

```
import android.os.Bundle;
import android.os.Handler;
import android.os.Looper;
import android.os.Message;
import android.util.Log;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.ProgressBar;
import android.widget.TextView;
import android.widget.Toast;
import androidx.appcompat.app.AppCompatActivity;
import com.example.ch06.R;
public class HandlerActivity extends AppCompatActivity implements OnClickListener {
    private static final String TAG = "HandlerActivity";
    private Handler countHandler = new Handler();
    private TextViewtvCount;
    private ProgressBarmProgressBar;
    private int count = 0;
    private Runnable mRunToast = new Runnable() {
        @Override
        public void run() {
            Toast.makeText(HandlerActivity.this, "15 秒后显示 Toast 提示信息",
            Toast.LENGTH_LONG).show();
        }
    };
    private Runnable mRunCount = new Runnable() {
        @Override
```

代码 5-4　HandlerActivity.java

```
        public void run() {
            //TODO Auto-generated method stub
            tvCount.setText("Count:" + String.valueOf(++count));
            countHandler.postDelayed(this, 1000);
        }
    };
    private Runnable mUpateProgressBarThread = new Runnable() {
        int i = 0;
        @Override
        public void run() {
            log("Begin Thread");
            i = i + 10;
            //得到一个消息对象,Message 类是由 Android 操作系统提供
            Message msg = updateProgressBarHandler.obtainMessage();
            //将 msg 对象的 arg1 参数的值设置为 i,用 arg1 和 arg2 这两个成员变量
            //传递消息,优点是系统性能消耗较少
            msg.arg1 = i;
            try {
                //设置当前显示睡眠 1s
                Thread.sleep(1000);
            } catch (InterruptedException e) {
                //TODO Auto-generated catch block
                e.printStackTrace();
            }
            //将 msg 对象加入到消息队列当中
            updateProgressBarHandler.sendMessage(msg);
            if (i == 100) {
                //如果当 i 的值为 100 时,就将线程对象从 handler 当中移除
                updateProgressBarHandler
                .removeCallbacks(mUpateProgressBarThread);
            }
        }
    };
    //使用匿名内部类来复写 Handler 当中的 handleMessage()方法
    private Handler updateProgressBarHandler =
                                    new Handler(Looper.getMainLooper()) {
        @Override
        public void handleMessage(Message msg) {
            mProgressBar.setProgress(msg.arg1);
            updateProgressBarHandler.post(mUpateProgressBarThread);
        }
    };
    @Override
    public void onClick(View view) {
        switch (view.getId()) {
            case R.id.btnStart:
                countHandler.postDelayed(mRunCount, 1000);
                break;
            case R.id.btnStop:
                countHandler.removeCallbacks(mRunCount);
                break;
```

代码 5-4 （续）

```
                case R.id.btnShowToast:
                    countHandler.postAtTime(mRunToast,
                    android.os.SystemClock.uptimeMillis() + 15 * 1000);
                    break;
                case R.id.btnUpdateProgressBar:
                    mProgressBar.setVisibility(View.VISIBLE);
                    updateProgressBarHandler.post(mUpateProgressBarThread);
                    break;
            }
        }
        @Override
        public void onCreate(Bundle savedInstanceState) {
            super.onCreate(savedInstanceState);
            setContentView(R.layout.c06_handler_test01);
            ((Button) findViewById(R.id.btnStart)).setOnClickListener(this);
            ((Button) findViewById(R.id.btnStop)).setOnClickListener(this);
            ((Button) findViewById(R.id.btnShowToast)).setOnClickListener(this);
            ((Button) findViewById(R.id.btnUpdateProgressBar))
            .setOnClickListener(this);
            mProgressBar = (ProgressBar) findViewById(R.id.progressBar1);
            tvCount = (TextView) findViewById(R.id.tvCount);
        }
        private void log(String msg) {
            Log.d(TAG, msg);
        }
    }
```

代码 5-4　（续）

（1）postDelayed（Runnable r，Object token，long delayMillis）：使 Runnable 添加到消息队列中，在经过指定的时间后运行。可运行对象将在附加此处理程序的线程上运行。时基是 SystemClock.uptimeMillis()。在深度睡眠中花费的时间会增加执行的额外延迟。

（2）removeCallbacks（Runnable r）：删除消息队列中所有待处理的 Runnable。

（3）postAtTime(Runnable r，long uptimeMillis)：使 Runnable 添加到消息队列中，在 uptimeMillis 给定的特定时间运行。时基是 SystemClock.uptimeMillis()。在深度睡眠中花费的时间会增加执行的额外延迟。可运行对象将在附加此处理程序的线程上运行。运行效果如图 5-3 所示。

图 5-3　HandlerActivity

5.2.3 AsyncTask

用 Handler 类在子线程中更新界面线程虽然避免了在主线程进行耗时计算，但费时的任务操作总会启动一些匿名的子线程，太多的子线程给系统带来巨大的负担，随之也带来一些性能问题。因此安卓提供了一个工具类 AsyncTask 来实现异步执行任务。AsyncTask 类擅于处理一些后台比较耗时的任务，给用户带来良好用户体验，不再需要子线程和 Handler 就可以完成异步操作并且刷新用户界面。如果要使用 AsyncTask，需要创建 AsyncTask 类，并实现其中的抽象方法以及重写某些方法。利用 AsyncTask 不需要自己来写后台线程，无须终结后台线程，但是 AsyncTask 的方式对循环调用的方式并不太合适。

AsyncTask 旨在使主线程能够正确和轻松地使用，然而最常见的用例是集成到界面中，这会导致上下文泄漏、错过回调或配置更改时崩溃。AsyncTask 在不同版本的平台上也有不一致的行为，doInBackground()会吞下异常，并且没有提供太多直接使用 Executor 的实用程序。AsyncTask 被设计为围绕 Thread 和 Handler 的辅助类，并不构成通用线程框架。AsyncTask 最适合用于短时间的操作（最多几秒钟）。如果需要保持线程长时间运行，强烈建议使用 java.util.concurrent 包提供的各种 API，例如，Executor、ThreadPoolExecutor 和 FutureTask。

异步任务由在后台线程上运行的计算定义，其结果在主线程上发布。异步任务由 3 个泛型定义，称为 Params、Progress 和 Result，以及四个步骤，称为 onPreExecute、doInBackground、onProgressUpdate 和 onPostExecute。AsyncTask 是抽象类，AsyncTask 定义了三种泛型：Params、Progress 和 Result，含义分别如下。

- Params：表示启动任务执行的输入参数，如 HTTP 请求的 URL。
- Progress：表示后台任务执行的百分比。
- Result：表示后台执行任务最终返回的结果，如 String、Integer 等。

通过继承一个 AsyncTask 类来定义一个异步任务类，安卓提供一个让程序员编写后台操作更为容易和透明的 AsyncTask，使得后台线程能够在界面主线程外进行处理。使用 AsyncTask，需要创建 AsyncTask 类，并实现其中的抽象方法以及重写某些方法。利用 AsyncTask 不需要自己来写后台线程，无须终结后台线程，AsyncTask 可实现多任务。下面的例子是实现进度条的更新，实现步骤（如代码 5-5 所示）如下。

（1）使用 execute()方法触发异步任务的执行。

（2）使用 onPreExecute()表示执行预处理，如绘制一个进度条控件。

（3）使用 doInBackground()执行较为费时的操作，这个方法是 AsyncTask 的关键，必须覆盖重写。

（4）使用 onProgressUpdate()对进度条控件根据进度值做出具体的响应。

（5）使用 onPostExecute()对后台任务的结果做出处理。

```
import android.os.AsyncTask;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
```

代码 5-5　AsyncTaskActivity.java