《移动终端开发技术》

电子教案

第一单元　Android 多线程编程

所属专业（教研室）：计算机软件技术

制定人：陈媛媛

合作人：

制定时间：2018 年 2 月

日照职业技术学院

<table>
<tr><td rowspan="2">单元标题</td><td rowspan="2">Android 多线程编程</td><td>单元教学学时</td><td>3 课时</td></tr>
<tr><td>在整体设计中的位置</td><td>第 23 次</td></tr>
<tr><td>授课班级</td><td></td><td>上课地点</td><td>一体化教室</td></tr>
<tr><td>上课时间</td><td colspan="3">周　　月　　日第　　节</td></tr>
<tr><td rowspan="2">教学
目标</td><td>能力目标</td><td>知识目标</td><td>素质目标</td></tr>
<tr><td>能够熟练运用异步消息处理机制、AsyncTask 进行多线程编程。</td><td>1、理解创建和使用线程的方法
2、掌握如何在子线程中更新 UI
3、掌握异步消息处理机制
4、掌握 AsyncTask 的使用方法</td><td>1、养成积极主动学习意识；
2、养成勤于动手的习惯。</td></tr>
<tr><td>教学重点、
难点</td><td colspan="3">教学重点：异步消息处理机制
教学难点：AsyncTask 的使用方法</td></tr>
<tr><td>教学
方法</td><td colspan="3">采用反转课堂教学模式，课前学生学习微课了解知识点，课上采用教师引导、演示，学生分组练习、讨论等教学方法。
运用多媒体、AndroidStudio 开发环境、实训助手、教学平台等辅助授课。</td></tr>
<tr><td>课前需掌握
的知识点</td><td colspan="3">Java 中创建和使用线程：
（1）继承 Thread 类实现多线程；
（2）实现 Runnable 接口的实现多线程</td></tr>
<tr><td>教学任务
分解</td><td colspan="3">任务一 创建和使用线程
任务二 在子线程中更新 UI
任务三 解析异步消息处理机制
任务四 使用 AsyncTask</td></tr>
<tr><td>教学
总结</td><td colspan="3"></td></tr>
</table>

教学过程

一、情景导入

老师直接说明，Android 中的四大组件我们已经学习完了 3 个，还差最后一个我们今天来学习。这个组件就是服务（Service），它能够长期在后台运行且不提供用户界面。即使用户切到另一应用程序，服务仍可以在后台运行。例如，使用音乐播放器播放音乐，将音乐播放器切换到后台仍然可以播放音乐。

需要注意的是服务并不是运行在一个独立的进程当中，而是依赖于创建服务时所在的应用程序进程。当某个应用程序进程被杀掉时，所有依赖该进程的服务也会停止运行。

服务并不会自动开启线程，所有的开发默认运行在主线程当中的，我们需要在服务的内部手动创建子线程，并在这里执行具体的任务，否则就有可能出现主线程被塞住的情况。

二、课前测试

1、Android 四大组件包括（ ）。

A Activity　　B service

C content provider　　D SQLite

答案 ABC

解析:Android 四大组件分别为 activity、service、content provider、broadcast receiver。

2、启动线程时需要调用的方法是( )。

A run()　　B start()

C execute()　　D create()

答案 B

3、Android 的异步消息处理机制中，发送消息一般使用 Handler 的（）方法。

A sendMessage（）　　B handleMessage（）

C loop（）　　D MessageQueue（）

答案 A

解析： handleMessage（）是对具体的 Message 进行处理。 loop（）是 Looper 的循环方法。MessageQueue（）不是方法，MessageQueue 是消息队列。

4、在使用 AsyncTask 时，（）方法中的所有代码都会在子线程中运行。

A onPreExecute()

B doInBackground(Params...)

C onProgressUpdate(Progress...)

D onPostExecute(Result)

答案 B

三、知识讲解

任务一、创建和使用线程

（1）继承 Thread 类实现多线程

Android 多线程编程和 Java 多线程编程基本都是使用相同的语法。需要新建一个类继承自 Thread，然后重写父类的 run()方法，并在里面编写耗时逻辑即可，如下所示：

```
  class MyThread extends Thread {
  @Override
public void run() {
// 处理具体的逻辑
}}
```

启动这个线程需要 new 出 MyThread 的实例，然后调用它的 start()方法，这样 run()方法中的代码就会在子线程当中运行了，如下所示：

```
new MyThread().start();
```

（2）实现 Runnable 接口的实现多线程

新建一个类实现 Runnable 接口，实现 run()方法。

```
class MyThread implements Runnable {
@Override
public void run() {
// 处理具体的逻辑
}}
```

启动线程的方法为：

```
MyThread myThread = new MyThread();
new Thread(myThread).start();
```

由于 Runnable 接口没有 start()方法，一个实现了 Runnable 接口的对象传入到 Thread 的构造函数里。接着调用 Thread 的 start()方法，run()方法中的代码就会在子线程当中运行了。也可以使用匿名类的方式， 如下所示：

```
new Thread(new Runnable() {
@Override
public void run() {
// 处理具体的逻辑
}
  }).start();
```

任务二、在子线程中更新 UI

和许多其他的 GUI 库一样，Android 的 UI 也是线程不安全的。也就是说，如果想要更新应用程序里的 UI 元素，则必须在主线程中进行，否则就会出现异常。

新建一个 AndroidThreadTest 项目，然后修改 activity_main.xml 中的代码，如下所示：

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android" androi
```

```
d:layout_width="match_parent"
android:layout_height="match_parent" >
<Button android:id="@+id/change_text"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:text="Change Text" />
<TextView android:id="@+id/text"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_centerInParent="true"
android:text="Hello world"
android:textSize="20sp" />
</RelativeLayout>
```

布局文件中定义了两个控件，TextView 用于在屏幕的正中央显示一个 Hello world 字符串，Button 用于改变 TextView 中显示的内容，我们希望在点击 Button 后可以把 TextView 中显示的字符串改成 Nice to meet you。

接下来修改 MainActivity 中的代码，如下所示：

```
public class MainActivity extends Activity implements OnClickListener {
private TextView text;
private Button changeText;
@Override
protected void onCreate(Bundle savedInstanceState)
{ super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);
text = (TextView) findViewById(R.id.text);
changeText = (Button) findViewById(R.id.change_text);
changeText.setOnClickListener(this);
}
@Override
public void onClick(View v) {
switch (v.getId()) {
case R.id.change_text:
new Thread(new Runnable() {
@Override
public void run() {
text.setText("Nice to meet you");}
}).start();
```

```
    break;
    default:
    break;
    }}}
```

可以看到，我们在 Change Text 按钮的点击事件里面开启了一个子线程，然后在子线程中调用 TextView 的 setText()方法将显示的字符串改成 Nice to meet you。代码的逻辑非常简单，只不过我们是在子线程中更新 UI 的。现在运行一下程序，并点击 Change Text 按钮，你会发现程序果然崩溃了，如图 10.1 所示。

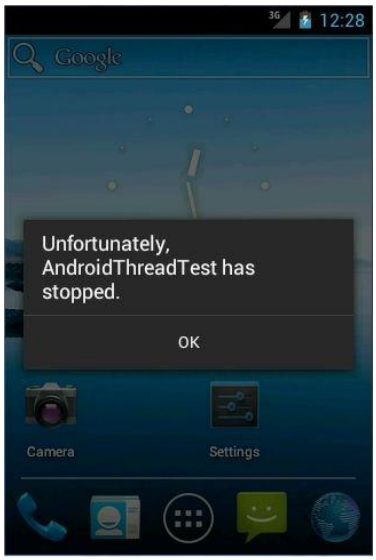


图 1 在子线程中更新 UI

然后观察 LogCat 中的错误日志，可以看出是由于在子线程中更新 UI 所导致的，如图 2 所示。

```
android.view.ViewRootImpl$CalledFromWrongThreadException: Only the original thread that
created a view hierarchy can touch its views.
```

图 2 错误

由此证实了 Android 确实是不允许在子线程中进行 UI 操作的。但是有些时候，我们必须在子线程里去执行一些耗时任务，然后根据任务的执行结果来更新相应的 UI 控件，这该如何是好呢？

对于这种情况，Android 提供了一套异步消息处理机制，完美地解决了在子线程中进行 UI 操作的问题。本小节中我们先来学习一下异步消息处理的使用方法，下一小节中再去分析它的原理。

修改 MainActivity 中的代码，如下所示：

```
    public class MainActivity extends Activity implements OnClickListener {
public static final int UPDATE_TEXT = 1;
```

```
private TextView text;
private Button changeText;
private Handler handler = new Handler() {
public void handleMessage(Message msg) {
switch (msg.what) {
case UPDATE_TEXT:
// 在这里可以进行 UI 操作
text.setText("Nice to meet you");
break;
default:
break;
}}  };
……
@Override
public void onClick(View v) {
switch (v.getId()) {
case R.id.change_text:
new Thread(new Runnable() {
@Override
public void run() {
Message message = new Message();
message.what = UPDATE_TEXT;
handler.sendMessage(message); // 将 Message 对象发送出去
}
}).start();
break;
default:
break;  }}}
```

这里我们先是定义了一个整型常量 UPDATE_TEXT，用于表示更新 TextView 这个动作。然后新增一个 Handler 对象，并重写父类的 handleMessage 方法，在这里对具体的 Message 进行处理。如果发现 Message 的 what 字段的值等于 UPDATE_TEXT，就将 TextView 显示的内容改成 Nice to meet you。

下面再来看一下 Change Text 按钮的点击事件中的代码。可以看到，这次我们并没有在子线程里直接进行 UI 操作，而是创建了一个 Message（android.os.Message）对象，并将它的 what 字段的值指定为 UPDATE_TEXT ，然后调用 Handler 的 sendMessage() 方法将这条 Message 发送出去。很快，Handler 就会收到这条 Message，并在 handleMessage()方法中对它进行处理。注意此时 handleMessage()方法中

的代码就是在主线程当中运行的了，所以我们可以放心地在这里进行 UI 操作。接下来对 Message 携带的 what 字段的值进行判断，如果等于 UPDATE_TEXT，就将 TextView 显示的内容改成 Nice to meet you。

现在重新运行程序，可以看到屏幕的正中央显示着 Hello world。然后点击一下 ChangeText 按钮，显示的内容着就被替换成 Nice to meet you，如图 10.3 所示。

图 3 子线程中更新 UI

这样你就已经掌握了 Android 异步消息处理的基本用法，使用这种机制就可以出色地解决掉在子线程中更新 UI 的问题。不过恐怕你对它的工作原理还不是很清楚，下面我们就来分析一下 Android 异步消息处理机制到底是如何工作的。

任务三、 解析异步消息处理机制

Android 中的异步消息处理主要由四个部分组成，Message、Handler、MessageQueue 和 Looper。下面我就对这四个部分进行一下简要的介绍。

（1）Message

Message 是在线程之间传递的消息，它可以在内部携带少量的信息，用于在不同线程之间交换数据。上一小节中我们使用到了 Message 的 what 字段，除此之外还可以使 用 arg1 和 arg2 字段来携带一些整型数据，使用 obj 字段携带一个 Object 对象。

（2） Handler

Handler 顾名思义也就是处理者的意思，它主要是用于发送和处理消息的。发送消息一般是使用 Handler 的 sendMessage()方法，而发出的消息经过一系列地辗转处理后， 最终会传递到 Handler 的 handleMessage()方法中。

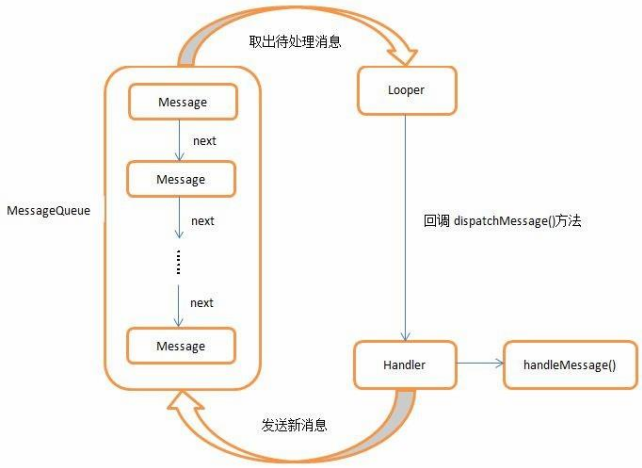
（3）MessageQueue

MessageQueue 是消息队列的意思，它主要用于存放所有通过 Handler 发送的消息。 这部分消息会一直存在于消息队列中，等待被处理。每个线程中只会有一个 MessageQueue 对象。

（4）Looper

Looper 是每个线程中的 MessageQueue 的管家，调用 Looper 的 loop()方法后，就会 进入到一个无限循环当中，然后每当发现 MessageQueue 中存在一条消息，就会将它取 出，并传递到 Handler 的 handleMessage()方法中。每个线程中也只会有一个 Looper 对象。

了解了 Message、Handler、MessageQueue 以及 Looper 的基本概念后，我们再来对异步消息处理的整个流程梳理一遍。首先需要在主线程当中创建一个 Handler 对象，并重写 handleMessage()方法。然后当子线程中需要进行 UI 操作时，就创建一个 Message 对象，并通过 Handler 将这条消息发送出去。之后这条消息会被添加到 MessageQueue 的队列中等待被处理，而 Looper 则会一直尝试从 MessageQueue 中取出待处理消息，最后分发回 Handler 的 handleMessage()方法中。由于 Handler 是在主线程中创建的，所以此时 handleMessage()方法中的代码也会在主线程中运行，于是我们在这里就可以安心地进行 UI 操作了。整个异步消息处理机制的流程示意图如图 4 所示。



一条 Message 经过这样一个流程的辗转调用后，也就从子线程进入到了主线程，从不能更新 UI 变成了可以更新 UI，整个异步消息处理的核心思想也就是如此。

任务四、使用 AsyncTask

不过为了更加方便我们在子线程中对 UI 进行操作，Android 还提供了另外一些好用的工具，AsyncTask 就是其中之一。借助 AsyncTask，即使你对异步消息处理机制完全不了解，也可以十分简单地从子线程切换到主线程。当然，AsyncTask 背后的实现原理也是基于异步消息处理机制的，只是 Android 帮我们做了很好的封装而已。

首先来看一下 AsyncTask 的基本用法，由于 AsyncTask 是一个抽象类，所以如果我们想使用它，就必须要创建一个子类去继承它。在继承时我们可以为 AsyncTask 类指定三个泛型参数，这三个参数的用途如下。

（1） Params

在执行 AsyncTask 时需要传入的参数，可用于在后台任务中使用。

（2）Progress

后台任务执行时，如果需要在界面上显示当前的进度，则使用这里指定的泛型作为进度单位。

（3） Result

当任务执行完毕后，如果需要对结果进行返回，则使用这里指定的泛型作为返回值类型。

因此，一个最简单的自定义 AsyncTask 就可以写成如下方式：

```
class DownloadTask extends AsyncTask<Void, Integer, Boolean> {
……
}
```

这里我们把 AsyncTask 的第一个泛型参数指定为 Void，表示在执行 AsyncTask 的时候不 需要传入参数给后台任务。第二个泛型参数指定为 Integer，表示使用整型数据来作为进度显 示单位。第三个泛型参数指定为 Boolean，则表示使用布尔型数据来反馈执行结果。

当然，目前我们自定义的 DownloadTask 还是一个空任务，并不能进行任何实际的操作， 我们还需要去重写 AsyncTask 中的几个方法才能完成对任务的定制。经常需要去重写的方法有以下四个。

（1）onPreExecute()

这个方法会在后台任务开始执行之前调用，用于进行一些界面上的初始化操作，比如显示一个进度条对话框等。

（2） doInBackground(Params...)

这个方法中的所有代码都会在子线程中运行，我们应该在这里去处理所有的耗时任务。任务一旦完成就可以通过 return 语句来将任务的执行结果返回，如果 AsyncTask 的 第三个泛型参数指定的是 Void，就可以不返回任务执行结果。注意，在这个方法中是不可以进行 UI 操作的，如果需要更新 UI 元素，比如说反馈当前任务的执行进度，可以调用 publishProgress(Progress...)方法来完成。

（3）onProgressUpdate(Progress...)

当在后台任务中调用了 publishProgress(Progress...)方法后，这个方法就会很快被调用，方法中携带的参数就是在后台任务中传递过来的。在这个方法中可以对 UI 进行操 作，利用参数中的数值就可以对界面元素进行相应地更新。

（4）onPostExecute(Result)

当后台任务执行完毕并通过 return 语句进行返回时，这个方法就很快会被调用。返 回的数据会作为参数传递到此方法中，可以利用返回的数据来进行一些 UI 操作，比如 说提醒任务执行的结果，以及关闭掉进度条对话框等。

因此，一个比较完整的自定义 AsyncTask 就可以写成如下方式：

```
class DownloadTask extends AsyncTask<Void, Integer, Boolean> {
```

```
    @Override
//后台任务执行前调用，进行界面上初始化操作
protected void onPreExecute() {
progressDialog.show(); // 显示进度对话框
}
  @Override
//所有代码在子线程中执行，处理耗时操作。执行具体下载任务。
protected Boolean doInBackground(Void... params) {
try {
while (true) {
int downloadPercent = doDownload(); // 这是一个虚构的方法
//将下载进度传进来，onProgressUpdate()方法很快被执行。
publishProgress(downloadPercent);
if (downloadPercent >= 100) {
break;
}
}
} catch (Exception e) {
return false;
}
return true;}
  @Override
//对 UI 进行操作，利用后台传递的参数对界面元素更新。
protected void onProgressUpdate(Integer... values) {
// 在这里更新下载进度
progressDialog.setMessage("Downloaded " + values[0] + "%");}
@Override
//后台任务执行完毕后，return 返回数据，调用该方法，利用返回的数据进行 UI 操作。
protected void onPostExecute(Boolean result) {
progressDialog.dismiss(); // 关闭进度对话框
// 在这里提示下载结果
if (result) {
Toast.makeText(context, "Download succeeded", Toast.LENGTH_SHORT).show();} else {
Toast.makeText(context, " Download failed", Toast.LENGTH_SHORT).show();
 }}}
```

在这个 DownloadTask 中，我们在 doInBackground()方法里去执行具体的下载任务。这个方法里的代码都是在子线程中运行的，因而不会影响到主线程的运行。注

	意这里虚构了一个 doDownload()方法，这个方法用于计算当前的下载进度并返回，我们假设这个方法已经存在了。在得到了当前的下载进度后，下面就该考虑如何把它显示到界面上了，由于 doInBackground()方法是在子线程中运行的，在这里肯定不能进行 UI 操作，所以我们可以调用 publishProgress()方法并将当前的下载进度传进来，这样 onProgressUpdate()方法就会很快被调用，在这里就可以进行 UI 操作了。 当下载完成后，doInBackground()方法会返回一个布尔型变量，这样 onPostExecute()方法就会很快被调用，这个方法也是在主线程中运行的。然后在这里我们会根据下载的结果来弹出相应的 Toast 提示，从而完成整个 DownloadTask 任务。 简单来说，使用 AsyncTask 的诀窍就是，在 doInBackground()方法中去执行具体的耗时任务，在 onProgressUpdate()方法中进行 UI 操作，在 onPostExecute()方法中执行一些任务的收尾工作。 如果想要启动这个任务，只需编写以下代码即可： new DownloadTask().execute(); 以上就是 AsyncTask 的基本用法，怎么样，是不是感觉简单方便了许多？我们并不需要去考虑什么异步消息处理机制，也不需要专门使用一个 Handler 来发送和接收消息，只需要调用一下 publishProgress()方法就可以轻松地从子线程切换到 UI 线程了。 **四、演示作品** 各小组演示汇报作品，教师与其他小组总结评价作品，各小组课后完善作品并提交到教学平台。 **五、知识巩固** 总结知识点，使用教学平台中的随堂练习题巩固本所学知识。
拓展作业	1、学习微课《定义一个服务》； 2、搜索资源，了解 Android 多线程的更多用法。
教学后记	