电子信息工程学院

移动终端开发技术

Android多线程编程

讲　师：陈媛媛

服务是什么

- **服务**（Service）是Android中实现程序后台运行的解决方案，它非常适合去执行那些不需要和用户交互而且还要求长期运行的任务。服务的运行不依赖任何用户界面。

需要注意的事项

服务是什么

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阻塞住的情况。

Service

1 多线程编程

知识构架

返回目录

7.1 Android多线程编程

- 7.1.1 线程的基本用法
- 7.1.2 在子线程中更新UI
- 7.1.3 解析异步消息处理机制
- 7.1.4 使用AsyncTask

线程的基本用法

在Java中如何实现多线程？

Android 多线程和 Java 多线程基本都是使用相同的语法

线程的基本用法

（1）继承Thread类实现多线程

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

启动这个线程：

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

（2）实现 Runnable 接口的实现多线程

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();
```

启动线程的方法为：

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

Android多线程的实现

在子线程中更新 UI

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

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。

案例：在子线程中更新UI

在子线程中更新 UI

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。

Handler 类（在主线程中）

重写handleMessage(Message msg)方法

{接收message.what字段

进行UI操作}

Message类（写在子线程中）

message.what = UPDATE_TEXT;

handler.sendMessage(message); // 将Message对象发送出去

更新UI的方法

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。

详细介绍一下这4部分吧

解析异步消息处理机制

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。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。

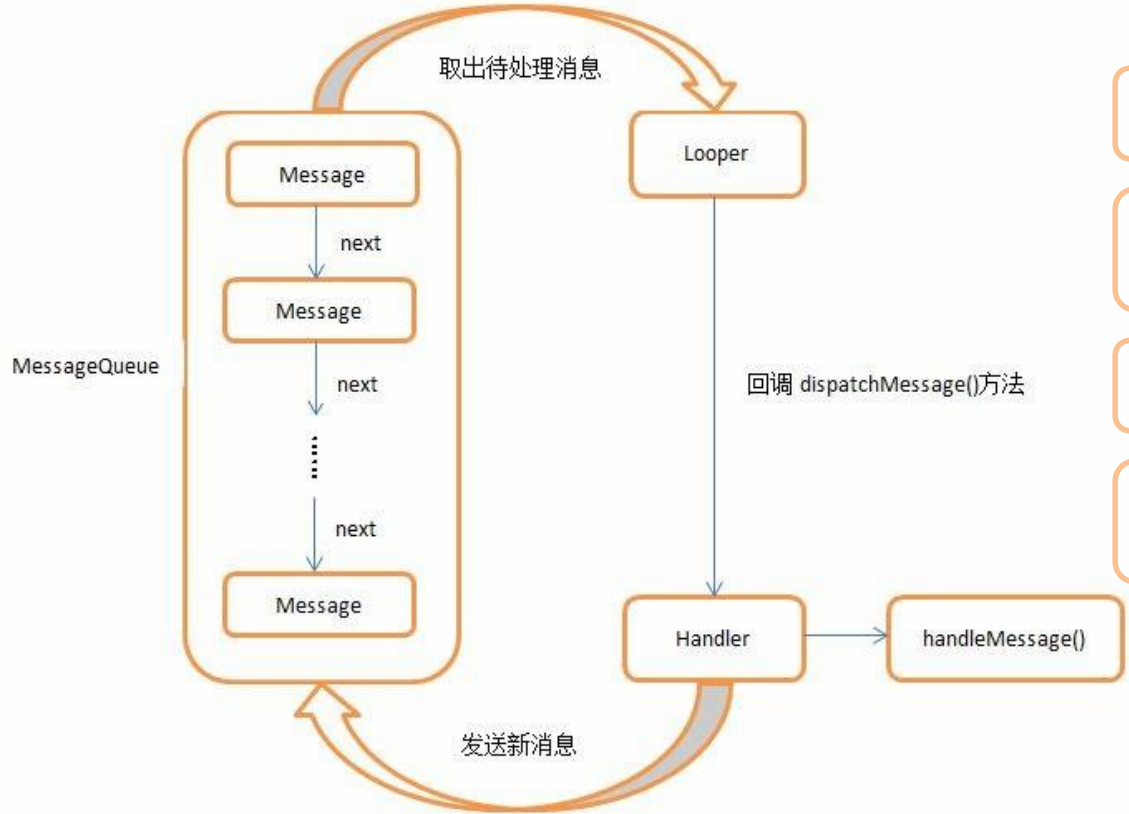
Handler 顾名思义也就是处理者的意思，它主要是用于发送和处理消息的。发送消息一般是使用 Handler 的 sendMessage()方法，而发出的消息经过一系列地辗转处理后， 最终会传递到 Handler 的 handleMessage()方法中。

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。

异步消息

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。

解析异步消息处理机制



首先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。

消息会被添加到 MessageQueue 的队列中等待被处理。

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。

使用 AsyncTask

AsyncTask 是一个抽象类，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。

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> {
……
}
```

重写 AsyncTask 中的几个方法才能完成任务的定制

使用 AsyncTask

重写4个方法

（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。

（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。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，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。

单元小结

1. 线程的基本用法
2. 在子线程中更新UI
3. 异步消息处理机制