Android移动应用开发基础教程

讲授：葛新

第8章 线程和服务

本章主要内容：

- 多线程
- 服务

8.1 多线程

在运行一个Android应用时，系统为其创建一个独立主线程。在程序执行一些比较耗时的操作（如打开网页）时，应用界面此时无法响应用户操作。将耗时操作放到子线程中去执行。子线程与主线程可以异步同时运行。当子线程去执行耗时操作时，用户可在界面中执行其他操作。

本节主要内容：

1. 线程的基本用法
2. 如何在使用多线程时更新UI
3. 使用AsyncTask

8.1.1 线程的基本用法

- 使用匿名类创建线程。
- 例如：

```
new Thread(new Runnable() {
    @Override
    public void run() {
        //在此编写线程功能代码
    }
}).start();
```

- new Thread()方法创建了一个线程对象，然后调用start()方法启动线程。new Runnable() {}创建了一个匿名类来实现Runnable接口，在其run()方法中编写实现线程功能的代码。

- 也可创建一个类来实现Runnable接口。例如：

```
class MyThread implements Runnable{
    @Override
    public void run() {
        //在此编写线程功能代码
    }
}
```

- 然后，按照下面的方式来启动线程。

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

- 也可定义一个类继承内置的Thread类来实现线程功能。例如：

```
class MyThread extends Thread{
    @Override
    public void run() {
        //在此编写线程功能代码
    }
}
```

- 然后，按照下面的方式来启动线程。

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

8.1.2 如何在使用多线程时更新UI

- 在Android中，不允许在主线程之外的子线程中修改应用界面。
- 例如，试图在子线程中将处理结果显示在Text View中，这样做会导致程序抛出异常。

```
private void showResult(final String result){
    runOnUiThread(new Runnable() {
        @Override
        public void run() {
            textView.setText(result);
        }
    });
}
```

- runOnUiThread()方法返回UI线程（也就是主线程）去执行，所以在其中设置TextView文本没有任何问题。

实例项目：源代码\08\UseThreadMessage

```
public class MainActivity extends AppCompatActivity {
    private Handler handler=new Handler(){
        @Override
        public void handleMessage(Message msg) {
            TextView textView=(TextView)findViewById(R.id.tvMsg);
            textView.setText(msg.obj.toString());
        }
    };
```

实例项目：源代码\08\UseThreadMessage

```
protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Button button=(Button)findViewById(R.id.button);
        button.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                new Thread(new Runnable() {
                    @Override
                    public void run() {
                        Message message=new Message();
                        message.obj=new String("线程中传回的数据");
                        handler.sendMessage(message);
                    }
                }).start();
```

……

Android中线程之间的消息传递也称异步消息处理机制，主要由Message、Handler、MessageQueue和Looper来完成。

1、Message：消息

- Message用于封装消息，它的arg1、arg2和what字段用于存放int类型数据，obj字段用于存放任意类型的对象。

2、Handler：消息处理器

- Handler主要用于发送和处理消息。通常，在子线程中调用sendMessage()方法发送消息。在主线程中执行handleMessage()方法处理消息。消息的发送和处理是异步执行的，不能期望消息发送之后，Handler能立即处理消息。

3、MessageQueue：消息队列

- 通过Handler发送的消息都保存在消息队列中，等待被处理。

4、Looper：消息循环

- Looper主要完成消息派遣任务。Looper维持一个无限循环，不停地检查消息队列中是否存在消息。当Looper发现消息队列中有消息时，就将队列最前面的消息取出，传递给Handler。

8.1.3 使用AsyncTask

- AsyncTask是Android为了简化使用线程数据更新UI而提供的一个抽象类。使用AsyncTask，不需要了解线程和异步消息处理机制，即可完成异步任务的执行。
- AsyncTask是一个抽象类，在使用时需要创建一个类来继承它

```
private class  MyAsyncTask extends AsyncTask<int[],String,String>{
        @Override
        protected void onPreExecute() {
            //异步任务开始执行之前执行的代码
        }
        @Override
        protected void onPostExecute(String s) {
            //异步任务执行结束之后执行的代码
        }
        @Override
        protected void onProgressUpdate(String... values) {
            //异步任务执行过程中执行的代码
        }
        @Override
        protected String doInBackground(int[]... params) {
            //异步任务代码
        }
    }
```

- 在继承AsyncTask时，首先需要指定3个泛型参数，其作用分别如下：
  - 第1个泛型参数：指定doInBackground()方法参数params的数据类型。参数params也称传入参数，保存调用AsyncTask子类构造函数时传入的参数。
  - 第2个泛型参数：指定onProgressUpdate ()方法参数values的数据类型。参数values保存在异步任务执行过程中传递回来的数据。
  - 第3个泛型参数：指定onPostExecute ()方法参数和doInBackground()方法返回值的数据类型。

- 此外，还需重写几个方法完成相应任务。
  - onPreExecute()方法：在异步任务开始执行之前被调用，并在主线程中运行。
  - onPostExecute ()方法：在异步任务执行结束之后被调用，并在主线程中运行。
  - onProgressUpdate ()方法：在异步任务代码中，可调用publishProgress ()方法向主线程返回数据，onProgressUpdate ()方法参数接收返回的数据。onProgressUpdate ()方法也在主线程中执行。
  - doInBackground()方法：异步任务代码，在子线程中执行。onPostExecute ()方法参数接收doInBackground()方法的返回值。

8.2 服务

通常，一个应用通过UI与用户进行交互。一些特殊的应用，例如与Web服务器的数据传输、下载文件、与服务器保持推送连接等，并不需要用户界面。这种应用就可使用服务来实现。

本节主要内容：

1. 使用服务
2. 使用绑定服务

8.2.1使用服务

- 创建的服务类

```
package com.example.xbg.useservice;
import android.app.Service;
import android.os.IBinder;
import android.content.Intent;
public class MyService extends Service {
    public MyService() {
    }
    @Override
    public IBinder onBind(Intent intent) {
        // TODO: Return the communication channel to the service.
        throw new UnsupportedOperationException("Not yet implemented");
    }
}
```

- 实现服务具体功能时，还需要重写Service的下列方法。

```
public void onCreate() {
    super.onCreate();
}
@Override
public int onStartCommand(Intent intent, int flags, int startId) {
    return super.onStartCommand(intent, flags, startId);
}
@Override
public void onDestroy() {
    super.onDestroy();
}
```

- 在调用startService()方法启动服务时，如果该服务还没有创建，则首先创建该服务，并执行onCreate()方法。如果服务已经创建，则不会执行onCreate()方法。注意，不管是在当前应用或其他应用中启动服务，服务的实例只有一个，onCreate()方法只执行一次。调用startService()方法启动服务时，如果服务已经创建，则执行onStartCommand()方法。每调用一次startService()方法，onStartCommand()方法就会执行一次。
- 服务启动后就会一直运行，调用stopService()方法（服务外调用）或stopSelf()方法（服务内）来停止服务。服务停止时，或执行onDestroy()方法。在调用了bindService()方法绑定了服务，然后调用unbindService()方法解除绑定时，也会执行onDestroy()方法。
- 从onCreate()方法到onDestroy()方法，经历服务的创建到销毁，是服务的一个完整生命周期。

- 在程序清单文件AndroidManifest.xml中添加服务注册消息。例如：

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.xbg.useservice">
    <application ..... >
        .....
        <service
            android:name=".MyService"
            android:enabled="true"
            android:exported="true"></service>
    </ application>
```

- 实现了服务类后，就可通过调用startService()方法启动服务。例如：
  startService(new Intent(MainActivity.this,MyService.class));
- 停止服务时，调用stopService()方法。例如。
  stopService(new Intent(MainActivity.this,MyService.class));
- 注意，在启动和停止服务时，虽然使用的是新建的Intent对象，但访问的是同一个服务，因为服务实例始终只有一个。

8.2.2 使用绑定服务

- 上一节中介绍的服务使用方法，可以称为服务的普通用法。在这种方式下，活动对服务控制只有启动和停止操作，服务中的代码如何执行与活动没有任何关系。
- Android提供了一种可以让活动和服务进行交互的方法——绑定服务。使用绑定服务，活动可以主动启动服务操作，并从服务返回数据。
- 在实现服务类时，onBind()方法返回一个IBinder对象，该对象通常是一个自定义的Binder子类的实例对象。通过IBinder对象，我们可以在活动中让任务完成指定操作。
- 要使用绑定类，首先需要实现服务类，并通过onBind()方法返回绑定对象。

实例项目：源代码\08\UseBindService

```
package com.example.xbg.usebindservice;
import android.app.Service;
……
public class MyService extends Service {
    public MyService() {}
    @Override
    public IBinder onBind(Intent intent) {   return new MyBinder();}//返回自定义绑定对象
    class MyBinder extends Binder{//自定义绑定类
        private int result=0;
        public void startDoSomething(int[] data){
            Log.e("MyService","MyBinder.startDoSomething()方法执行...");
            for(int i=0;i<data.length;i++) result+=data[i];
        }
        public int getResult(){return  result;}
    }
}
```

实例项目：源代码\08\UseBindService

```
package com.example.xbg.usebindservice;
import android.content.ComponentName;
……
public class MainActivity extends AppCompatActivity implements View.OnClickListener  {
    class MyServiceConnection implements ServiceConnection{
        @Override
        public void onServiceConnected(ComponentName name, IBinder service) {
            Log.e("MainActivity","服务绑定完成");
            MyService.MyBinder myBinder= (MyService.MyBinder) service;
            myBinder.startDoSomething(new int[]{1,2,3,4,5});
            Log.e("MainActivity","服务返回数据："+ myBinder.getResult());
        }
        @Override
        public void onServiceDisconnected(ComponentName name) {}
    }
    private MyServiceConnection myConnection=new MyServiceConnection();
```

实例项目：源代码\08\UseBindService

```
protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        findViewById(R.id.btBindService).setOnClickListener(this);
        findViewById(R.id.btUnBindService).setOnClickListener(this);
    }
```

实例项目：源代码\08\UseBindService

```
public void onClick(View v) {
    switch(v.getId()){
        case R.id.btBindService://执行绑定服务操作
            Intent intent=new Intent(this,MyService.class);
            bindService(intent,myConnection,BIND_AUTO_CREATE);
            break;
        case R.id.btUnBindService://执行解除绑定操作
            unbindService(myConnection);
            break;
    }
}
```