**异步更新之Handler.post解惑**

对于线程的控制，我们将介绍一个 Handler类，使用该类可以对运行在不同线程中的多个任务进行排队，并使用Message和Runnable对象安排这些任务。在javadoc中，对Handler是这样解释的：Handler可以发送和处理消息对象或Runnable对象，这些消息对象和Runnable对象与一个线程相关联。每个Handler的实例都关联了一个线程和线程的消息队列。当创建了一个Handler对象时，一个线程或消息队列同时也被创建，该Handler对象将发送和处理这些消息或Runnable对象。  
  
下面有几种对Handler对象的构造方法需要了解一下：  
  
**a、如果new一个无参构造函数的Handler对象，那么这个Handler将自动与当前运行线程相关联，也就是说这个Handler将与当前运行的线程使用同一个消息队列，并且可以处理该队列中的消息。**

private Handler handler = new Handler();

　　我们做这样一个实验，在主用户界面中创建一个带有无参构造函数的Handler对象，该Handler对象向消息队列推送一个Runnable对象，在Runnable对象的run函数中打印当前线程Id，我们比较主用户界面线程ID和Runnable线程ID是否相同。具体代码如下：

public class HandlerTest01 extends Activity

{

 @Override

public void onCreate(Bundle savedInstanceState){

super.onCreate(savedInstanceState);

setContentView(R.layout.main);

System.out.println("Activity ---> " + Thread.currentThread().getId());

handler.post(r);

 }

private Handler handler = new Handler();

private Runnable r = new Runnable(){

@Override

public void run() {

try {

Thread.sleep(2000);

} catch (InterruptedException e) {

e.printStackTrace();

}

  System.out.println("Runnalbe ---> " + Thread.currentThread().getId());

}

};

}

　　通过这个例子的输出可以发现，Runnable对象和主用户界面线程的ID是相同。在这个例子中，我们直接利用handler对象post了一个runnable对象，相当于直接调用了Runnable对象的run函数，也就说没有经过start函数调用run()，那么就不会创建一个新线程，而是在原有线程内部直接调用 run()方法，因此输出的线程Id是相同的。  
  
**b、如果new一个带参构造函数的Handler对象，那么这个Handler对象将与参数所表示的Looper相关联。**注意：此时线程类应该是一个特殊类HandlerThread类，一个Looper类的Thread类，它继承自Thread类。

HandlerThread handlerthread = new HandlerThread("MyThread");

handlerthread.start();

private MyHandler handler = new MyHandler(handlerthread.getLooper());

class MyHandler extends Handler

{

public MyHandler() { }

public MyHandler(Looper looper)

 {

super(looper);

 }

 }

　　下面这个例子，将介绍如何开启一个新的线程，并通过Handler处理消息。

public class HandlerTest02 extends Activity

{

private MyHandler myhandler = null;

 @Override

protected void onCreate(Bundle savedInstanceState)

{

super.onCreate(savedInstanceState);

this.setContentView(R.layout.main);

 System.out.println("Activity ---> " + Thread.currentThread().getId());

// 生成一个HandlerThread对象，使用Looper来处理消息队列

HandlerThread thread = new HandlerThread("MyThread");

// 必须启动这个线程

thread.start();

// 将一个线程绑定到Handler对象上，则该Handler对象就可以处理线程的消息队列

myhandler = new MyHandler(thread.getLooper());

// 从Handler中获取消息对象

Message msg = myhandler.obtainMessage();

// 将msg对象发送给目标对象Handler

msg.sendToTarget();

 }

class MyHandler extends Handler

{

public MyHandler() { }

// 带有参数的构造函数

public MyHandler(Looper looper)

{

super(looper);

}

 @Override

public void handleMessage(Message msg)

{

System.out.println("MyHandler ---> " + Thread.currentThread().getId());

 }

 }

 }

　　根据这个例子返回的结果，可以看出，新线程Id与主用户界面的线程Id不同。由于我们调用了thread.start()方法，真正的创建了一个新线程，与原来的线程处于不同的线程上下文中，因此打印输出的线程Id是不同的。  
**c、如果需要Handler对象去处理消息，那么就要重载Handler类的handleMessage函数。**

private Handler handler = new Handler() {

@Override

public void handleMessage(Message msg)

 {

// TODO : Handle the msg

// Usually we update UI here.

}

}

　　注意到注释部分，我们通常在handleMessage中处理更新UI界面的操作。  
  
　　前面介绍了Handler类的基本使用，但是还是没有涉及到Thread类。要想实现在后台重新开启一个新的线程，通过该线程执行一些费时的操作，我们也使用Thread类来完成这个功能。下面我们先给出一个使用Thread类的例子程序。

public class ThreadTest extends Activity

{

@Override

protected void onCreate(Bundle savedInstanceState){

super.onCreate(savedInstanceState);

this.setContentView(R.layout.main);

System.out.println("Activity ---> " + Thread.currentThread().getId());

Thread thread = new Thread(r);

thread.start();

try {

Thread.currentThread().sleep(1000);

} catch (InterruptedException e){

e.printStackTrace();

}

thread.stop();

 }

Runnable r = new Runnable() {

@Override

public void run()

{

System.out.println("Runnable ---> " + Thread.currentThread().getId());

}

};

}

　　这个程序执行的结果如下。新线程在创建对象时，传入了Runnable类的一个对象，在Runnable对象中重载了run()方法去执行耗时的操作；新的线程实例执行了start方法，开启了一个新的线程执行Runnable的run方法。  
  
上面这些就是我现在接触到执行线程的方法，在线程中，可以完成我们所需要的操作(比如：下载，处理数据，检测网络状态等)，使其与UI界面分离，那么UI界面不会因为耗时操作导致界面被阻塞。  
  
　　在《解密Google Android》一书中，发现了这样一个启动线程的模型。利用该模型，我们可以把一些耗时的操作放到doStuff方法中去执行，同时在 updateUIHere方法中进行更新UI界面的操作，就可以完成一个线程所需要的功能。其他的说明写在注释部分了。

Handler myHandler = new Handler() {

public void handleMessage(Message msg){

  updateUIHere();

}

}

new Thread() {

public void run()

{

 doStuff();

// 执行耗时操作

Message msg = myHandler.obtainMessage();

 Bundle b = new Bundle();

 b.putString("key", "value");

m.setData(b);

// 向消息中添加数据

myHandler.sendMessage(m);

// 向Handler发送消息，更新UI

}

}.start();

=====================================================================================  
通过上面的文章，可以对handler的处理机制有一个比较深刻的认识，而我之前的疑问，为什么在handler中post一个线程，也就有了答案。  
handler的应用相当一个对多线程管理的机制，通过一个线程的队列（因为可以post多个）来管理这些线程，  
同时通过使用Handler或者HandlerThread来控制，这些线程是要在主线程中执行还是要新开一个线程来执行。

public class HandlerActivity extends Activity

{

@Override

public void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.main);

Button b1 = (Button)findViewById(R.id.button1);

Button b2 = (Button)findViewById(R.id.button2);

b1.setOnClickListener(new OnClickListener() {

@Override

public void onClick(View v)

{

//立即把线程加入消息队列中

handler.post(r);

} });

b2.setOnClickListener(new OnClickListener() {

@Override

public void onClick(View v)

{

//停止线程

handler.removeCallbacks(r);

}});

}

Handler handler = new Handler();

/\*\* \*该方法的内部类将在handler.sendMessage(msg)后执行

Handler handler = new Handler(){

@Override

public void handleMessage(Message msg)

{

 System.out.println("msg:"+msg.arg1);

}

};

\*/

Runnable r = new Runnable()

{

@Override

public void run()

{

System.out.println("sysout Thread");

//得到一个消息对象

Message msg = handler.obtainMessage();

//将msg对象arg1参数设置为122，用arg1和arg2传递消息，优点是系统资源消耗较小

msg.arg1 = 122;

//将消息加入到另外一个消息队列中去

handler.sendMessage(msg);

//3000毫秒后加入线程到消息队列中

handler.postDelayed(r, 3000);

}

};

}