# UI操作的原因

handler是Android给我们提供用来更新UI的一套机制，也是一套消息处理机制，我们可以发消息，也可以通过它处理消息。

当我们需要在子线程处理耗时的操作（例如访问网络，数据库的操作），而当耗时的操作完成后，需要更新UI，这就需要使用Handler来处理，因为子线程不能做更新UI的操作。UI主线程不允许其他线程操作组件，只有UI主线程能够绘制界面。

Handler能帮我们很容易的把任务（在子线程处理）切换回它所在的线程。简单理解，Handler就是解决线程和线程之间的通信的。

# 线程之间通过Handler通信

## 子线程发送消息给主线程

子线程发送消息给主线程：由于主线程中本身就有Looper机制，里面有消息队列，所以子线程只需要获得主线程中的Handler对象，用它发送消息，主线程中的Looper会查看到排队的message。

范例：倒计时；

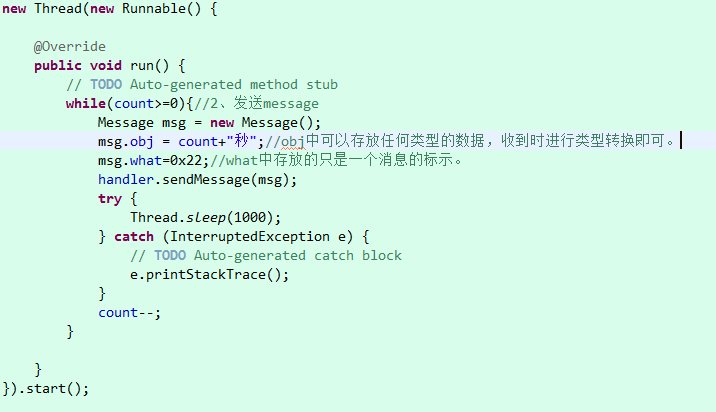
功能：点击按钮，开始60秒倒计时，时间再textview显示，结束时，按钮显示“计时结束”。

思路：这里是子线程发送消息给主线程。

发送的是count秒数，主线程接收到消息，绘制界面显示到textview中。



创建主线程中Handler的子类，重写其handleMessage()方法，接收消息进行处理。what判断消息标识，可以没有，反正都会接收到。



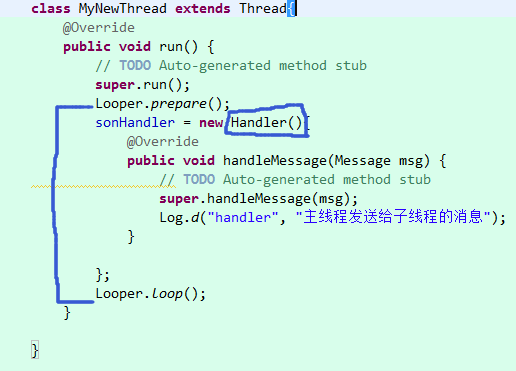
创建的子线程中，利用主线程中的handler进行消息发送，设置标识在what中，传入消息在obj中，可以存放任意类型的消息。

## 主线程发送消息给子线程

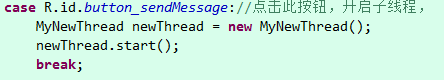
主线程发送消息给子线程：子线程中没有Looper机制，所以需要创建Looper和Handler，然后主线程用子线程中的Handler实例进行发送消息，子线程就能收到。

功能：主线程发送消息，子线程handler收到打出Log

创建子线程，并创建其Looper机制：



开启线程：



点击按钮，发送消息：

sonHandler.sendEmptyMessage(0); //获得子线程的sonHandler。向开启的子线程中发消息。

注意这两个方法！

Looper.prepare();

Looper.loop();

# Handler中涉及的内存泄漏

Handler导致的内存泄漏的问题，同AsyncTask是一样的原因。

使用Handler导致内存泄露的解决方法

方法一：通过程序逻辑来进行保护。

1.在关闭Activity的时候停掉你的后台线程。线程停掉了，就相当于切断了Handler和外部连接的线，Activity自然会在合适的时候被回收。

2.如果你的Handler是被delay的Message持有了引用，那么使用相应的Handler的removeCallbacks()方法，把消息对象从消息队列移除就行了。

方法二：将Handler声明为静态类 + 弱引用

PS:在Java 中，非静态的内部类和匿名内部类都会隐式地持有其外部类的引用，静态的内部类不会持有外部类的引用。

静态类不持有外部类的对象，所以你的Activity可以随意被回收。由于Handler不再持有外部类对象的引用，导致程序不允许你在Handler中操作Activity中的对象了。所以你需要在Handler中增加一个对Activity的弱引用（WeakReference）。

PS：什么是WeakReference？

　　WeakReference弱引用，与强引用（即我们常说的引用）相对，它的特点是，GC在回收时会忽略掉弱引用，即就算有弱引用指向某对象，但只要该对象没有被强引用指向（实际上多数时候还要求没有软引用，但此处软引用的概念可以忽略），该对象就会在被GC检查到时回收掉。对于上面的代码，用户在关闭Activity之后，就算后台线程还没结束，但由于仅有一条来自Handler的弱引用指向Activity，所以GC仍然会在检查的时候把Activity回收掉。这样，内存泄露的问题就不会出现了。

static class MyHandler extends Handler{

private final WeakReference<MyActivity> mActivity;

public MyHandler(MyActivity activity){

super();

mActivity = new WeakReference<MyActivity>(activity);

}

@Override

public void handleMessage(Message msg) {

MyActivity myActivity = mActivity.get();

if (myActivity!=null){

myActivity.textView.setText("123456789");

}

}

}