# **Android中Handler引起的内存泄露**

在Android常用编程中，Handler在进行异步操作并处理返回结果时经常被使用。通常我们的代码会这样实现。

public class SampleActivity extends Activity {  
 private final Handler mLeakyHandler = new Handler() {  
 @Override  
 public void handleMessage(Message msg) {  
 }  
 };  
}

但是，其实上面的代码可能导致内存泄露，当你使用Android lint工具的话，会得到这样的警告

*In Android, Handler classes should be static or leaks might occur, Messages enqueued on the application thread’s MessageQueue also retain their target Handler. If the Handler is an inner class, its outer class will be retained as well. To avoid leaking the outer class, declare the Handler as a static nested class with a WeakReference to its outer class*

看到这里，可能还是有一些搞不清楚，代码中哪里可能导致内存泄露，又是如何导致内存泄露的呢？那我们就慢慢分析一下。

1.当一个Android应用启动的时候，会自动创建一个供应用主线程使用的Looper实例。Looper的主要工作就是一个一个处理消息队列中的消息对象。在Android中，所有Android框架的事件（比如Activity的生命周期方法调用和按钮点击等）都是放入到消息中，然后加入到Looper要处理的消息队列中，由Looper负责一条一条地进行处理。主线程中的Looper生命周期和当前应用一样长。

2.当一个Handler在主线程进行了初始化之后，我们发送一个target为这个Handler的消息到Looper处理的消息队列时，实际上已经发送的消息已经包含了一个Handler实例的引用，只有这样Looper在处理到这条消息时才可以调用Handler#handleMessage(Message)完成消息的正确处理。

3.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。

确实上面的代码示例有点难以察觉内存泄露，那么下面的例子就非常明显了

public class SampleActivity extends Activity {  
 private final Handler mLeakyHandler = new Handler() {  
 @Override  
 public void handleMessage(Message msg) {  
 }  
 };  
  
 @Override  
 protected void onCreate(Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
 // Post a message and delay its execution for 10 minutes.   
 mLeakyHandler.postDelayed(new Runnable() {  
 @Override  
 public void run() {  
 }  
 }, 1000 \* 60 \* 10);  
 // Go back to the previous Activity.   
 finish();  
 }  
}

分析一下上面的代码，当我们执行了Activity的finish方法，被延迟的消息会在被处理之前存在于主线程消息队列中10分钟，而这个消息中又包含了Handler的引用，而Handler是一个匿名内部类的实例，其持有外面的SampleActivity的引用，所以这导致了SampleActivity无法回收，进行导致SampleActivity持有的很多资源都无法回收，这就是我们常说的内存泄露。

注意上面的new Runnable这里也是匿名内部类实现的，同样也会持有SampleActivity的引用，也会阻止SampleActivity被回收。

要解决这种问题，思路就是避免使用非静态内部类，继承Handler时，要么是放在单独的类文件中，要么就是使用静态内部类。因为静态的内部类不会持有外部类的引用，所以不会导致外部类实例的内存泄露。当你需要在静态内部类中调用外部的Activity时，我们可以使用[弱引用](http://droidyue.com/blog/2014/10/12/understanding-weakreference-in-java/" \o "弱引用" \t "http://droidyue.com/blog/2014/12/28/in-android-handler-classes-should-be-static-or-leaks-might-occur/_self)来处理。另外关于同样也需要将Runnable设置为静态的成员属性。注意：一个静态的匿名内部类实例不会持有外部类的引用。 修改后不会导致内存泄露的代码如下

public class SampleActivity extends Activity {  
 */\*\*  
 \* Instances of static inner classes do not hold an implicit reference to their outer class.  
 \*/* private static class MyHandler extends Handler {  
 private final WeakReference<SampleActivity> mActivity;  
  
 public MyHandler(SampleActivity activity) {  
 mActivity = new WeakReference<SampleActivity>(activity);  
 }  
  
 @Override  
 public void handleMessage(Message msg) {  
 SampleActivity activity = mActivity.get();  
 if (activity != null) {  
 }  
 }  
 }  
  
 private final MyHandler mHandler = new MyHandler(this);  
 */\*\*  
 \* Instances of anonymous classes do not hold an implicit reference to their outer class when they are "static".  
 \*/* private static final Runnable *sRunnable* = new Runnable() {  
 @Override  
 public void run() {  
 }  
 };  
  
 @Override  
 protected void onCreate(Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
 // Post a message and delay its execution for 10 minutes.   
 mHandler.postDelayed(*sRunnable*, 1000 \* 60 \* 10); // Go back to the previous Activity.   
 finish();  
 }  
}

其实在Android中很多的内存泄露都是由于在Activity中使用了非静态内部类导致的，就像本文提到的一样，所以当我们使用时要非静态内部类时要格外注意，如果其实例的持有对象的生命周期大于其外部类对象，那么就有可能导致内存泄露。个人倾向于使用文章的静态类和弱引用的方法解决这种问题。