Handler内存泄漏分析及解决

一、介绍

首先，请浏览下面这段handler代码：

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

在使用handler时，这是一段很常见的代码。但是，它却会造成严重的内存泄漏问题。在实际编写中，我们往往会得到如下警告：

```
⚠ In Android, Handler classes should be static or leaks might occur.
```

二、分析

1、 Android角度

当Android应用程序启动时，framework会为该应用程序的主线程创建一个Looper对象。这个Looper对象包含一个简单的消息队列Message Queue，并且能够循环的处理队列中的消息。这些消息包括大多数应用程序framework事件，例如Activity生命周期方法调用、button点击等，这些消息都会被添加到消息队列中并被逐个处理。

另外，主线程的Looper对象会伴随该应用程序的整个生命周期。

然后，当主线程里，实例化一个Handler对象后，它就会自动与主线程Looper的消息队列关联起来。所有发送到消息队列的消息Message都会拥有一个对Handler的引用，所以当Looper来处理消息时，会据此回调[Handler#handleMessage(Message)]方法来处理消息。

2、 Java角度

在java里，非静态内部类 和 匿名类 都会潜在的引用它们所属的外部类。但是，静态内部类却不会。

三、泄漏来源

请浏览下面一段代码：

```
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();
  }
}
```

当activity结束(finish)时，里面的延时消息在得到处理前，会一直保存在主线程的消息队列里持续10分钟。而且，由上文可知，这条消息持有对handler的引用，而handler又持有对其外部类（在这里，即SampleActivity）的潜在引用。这条引用关系会一直保持直到消息得到处理，从而，这阻止了SampleActivity被垃圾回收器回收，同时造成应用程序的泄漏。

<<<<<<< HEAD
注意，上面代码中的Runnable类--非静态匿名类--同样持有对其外部类的引用。从而也导致泄漏。

四、泄漏解决方案

首先，上面已经明确了内存泄漏来源：

只要有未处理的消息，那么消息会引用handler，非静态的handler又会引用外部类，即Activity，导致Activity无法被回收，造成泄漏；

Runnable类属于非静态匿名类，同样会引用外部类。

为了解决遇到的问题，我们要明确一点：静态内部类不会持有对外部类的引用。所以，我们可以把handler类放在单独的类文件中，或者使用静态内部类便可以避免泄漏。

另外，如果想要在handler内部去调用所在的外部类Activity，那么可以在handler内部使用弱引用的方式指向所在Activity，这样统一不会导致内存泄漏。

对于匿名类Runnable，同样可以将其设置为静态类。因为静态的匿名类不会持有对外部类的引用。

```
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();
    }
}
```

五、小结

虽然静态类与非静态类之间的区别并不大，但是对于Android开发者而言却是必须理解的。至少我们要清楚，如果一个内部类实例的生命周期比Activity更长，那么我们千万不要使用非静态的内部类。最好的做法是，使用静态内部类，然后在该类里使用弱引用来指向所在的Activity。