Android内存泄露之Handler

泡在网上的日子 发表于 2014-11-22 20:29 第 887 次阅读 内存

0

来源 http://blog.csdn.net/zhuanglonghai/article/details/38233069

Thread 内存泄露

线程也是造成内存泄露的一个重要的源头。线程产生内存泄露的主要原因在于线程生命周期的不可控。

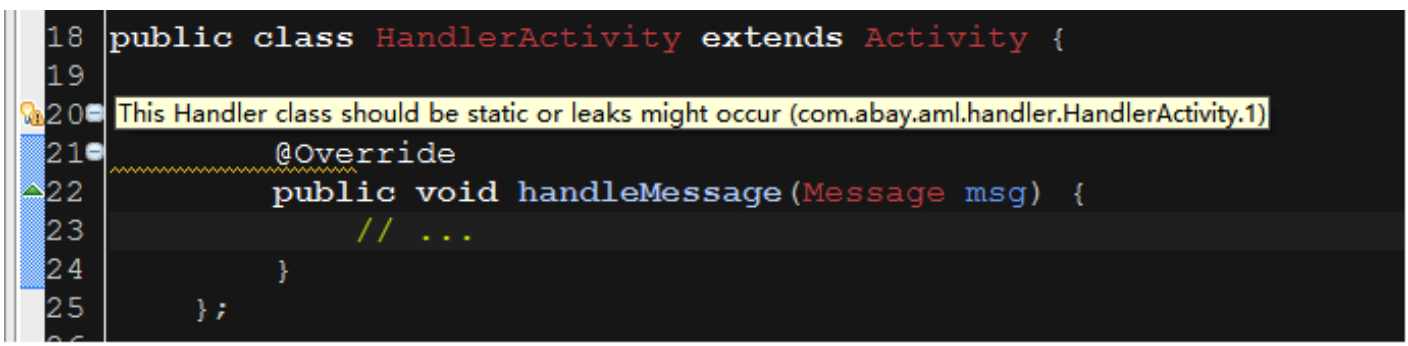
1.看一下下面是否存在问题

```
/**
 *
 * @version 1.0.0
 * @author Abay Zhuang <br/>
 *         Create at 2014-7-17
 */
public class ThreadActivity extends Activity {
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        new MyThread().start();
    }
    private class MyThread extends Thread {
        @Override
        public void run() {
            super.run();
            dosomthing();
        }
    }
    private void dosomthing(){

    }
}

```

是否您以前也是这样用的呢。

没有问题?

Eclipse 工具有这样的警告提示 警告:




```
This Handler class should be static or leaks might occur
(com.example.ta.HandlerActivity.1) 意思：class 使用静态声明否者可能
```

为啥出现这样的问题呢

Handler 的生命周期与Activity 不一致

当Android应用启动的时候，会先创建一个UI主线程的Looper对象，Looper实现了一个简单的消息队列，一个一个的处理里面的Message对象。主线程Looper对象在整个应用生命周期中存在。

当在主线程中初始化Handler时，该Handler和Looper的消息队列关联（没有关联会报错的）。发送到消息队列的Message会引用发送该消息的Handler对象，这样系统可以调用Handler#handleMessage(Message) 来分发处理该消息。

handler 引用 Activity 阻止了GC对Acivity的回收

在Java中，非静态(匿名)内部类会默认隐性引用外部类对象。而静态内部类不会引用外部类对象。

如果外部类是Activity，则会引起Activity泄露 。

当Activity finish后，延时消息会继续存在主线程消息队列中1分钟，然后处理消息。而该消息引用了Activity的Handler对象，然后这个Handler又引用了这个Activity。这些引用对象会保持到该消息被处理完，这样就导致该Activity对象无法被回收，从而导致了上面说的 Activity泄露。

如何避免?

使用显形的引用，1.静态内部类。 2. 外部类

使用弱引用 2. WeakReference

修改代码如下:

```
/**
 *
 * 实现的主要功能。
 *
 * @version 1.0.0
 * @author Abay Zhuang <br/>
 *         Create at 2014-7-28
 */
public class HandlerActivity2 extends Activity {
    private static final int MESSAGE_1 = 1;
    private static final int MESSAGE_2 = 2;
    private static final int MESSAGE_3 = 3;
    private final Handler mHandler = new MyHandler(this);
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        mHandler.sendMessageDelayed(Message.obtain(), 60000);
        // just finish this activity
        finish();
    }
    public void todo() {
    };
    private static class MyHandler extends Handler {
        private final WeakReference<HandlerActivity2> mActivity
        public MyHandler(HandlerActivity2 activity) {
            mActivity = new WeakReference<HandlerActivity2>(act
        }
        @Override
        public void handleMessage(Message msg) {
            System.out.println(msg);
            if (mActivity.get() == null) {
                return;
            }
```

```
            mActivity.get().todo();
        }
    }
```

上面这样就可以了吗?

当Activity finish后 handler对象还是在Message中排队。 还是会处理消息，这些处理有必要？ 正常Activitiy finish后，已经没有必要对消息处理，那需要怎么做呢?

解决方案也很简单，在Activity onStop或者onDestroy的时候，取消掉该Handler对象的Message和Runnable。 通过查看Handler的API，它有几个方法:

removeCallbacks(Runnable r)和removeMessages(int what)等。

代码如下:

```
/**
 * 一切都是为了不要让mHandler拖泥带水
 */
@Override
public void onDestroy() {
    mHandler.removeMessages(MESSAGE_1);
    mHandler.removeMessages(MESSAGE_2);
    mHandler.removeMessages(MESSAGE_3);
    // ... ...
    mHandler.removeCallbacks(mRunnable);
    // ... ...
}
```

如果上面觉的麻烦，也可以如下面:

```
@Override
public void onDestroy() {
    //  If null, all callbacks and messages will be removed.
    mHandler.removeCallbacksAndMessages(null);
}
```