



作者 BlackSwift (/users/b99b0edd4e77) 2015.03.09 23:44\*

写了68985字，被471人关注，获得了775个喜欢  
(/users/b99b0edd4e77)

+ 添加关注 (/sign\_in)

# 使用Android studio分析内存泄露

字数1340 阅读14732 评论19 喜欢78

This post is a permitted translation of badoo Tech Blog (<https://techblog.badoo.com/blog/2014/08/28/android-handler-memory-leaks/>) and I add some text and screenshots for android studio users.

Origin Author: Dmytro Voronkevych  
(<https://techblog.badoo.com/authors/dmytro-voronkevych/>)

follow badoo on Tweet (<https://twitter.com/badoootech>)

Translator: Miao1007

截至androidstudio1.3为止，其内部的MemoryDump功能都很难使用，还是使用MAT更佳。

Android使用java作为平台开发，帮助我们解决了很多底层问题，比如内存管理，平台依赖等等。然而，我们也经常遇到 `OutOfMemory` 问题，垃圾回收到底去哪了？

接下来是一个 `Handler Leak` 的例子，它一般会在编译器中被警告提示。

## 所需要的工具

- Android Studio 1.1 or higher
- Eclipse MemoryAnalyzer

## 示例代码

```
public class NonStaticNestedClassLeakActivity extends ActionBarActivity {

    TextView textView;

    public static final String TAG = NonStaticNestedClassLeakActivity.class.getSimpleName()

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_non_static_nested_class_leak);
        textView = (TextView)findViewById(R.id.textview);
        Handler handler = new Handler();

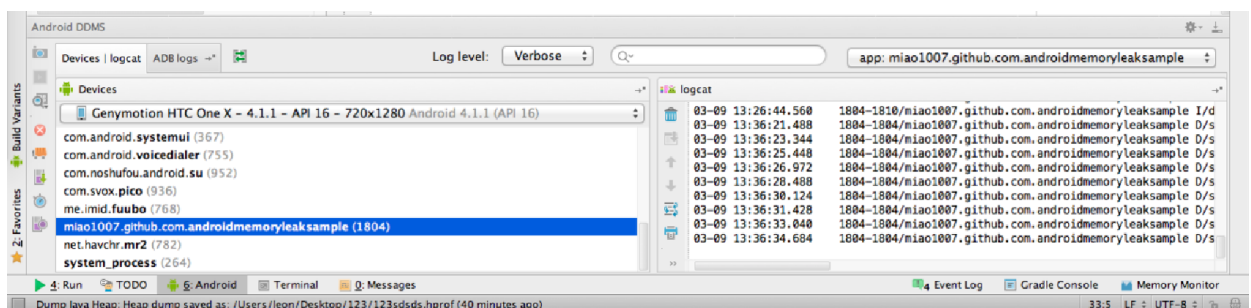
        handler.postDelayed(new Runnable() {
            @Override public void
                textView.setText("Done");
            }//a mock for long time work
        }, 800000L);

    }
}
```

这是一个非常基础的Activity.注意这个匿名的 Runnable 被送到了Handler中，而且延迟非常的长。现在我们运行这个Activity,反复旋转屏幕，然后导出内存并分析。

## 导入 Memory 到Eclipse MemoryAnalyzer

### 使用Androidstudio导出 heap dump



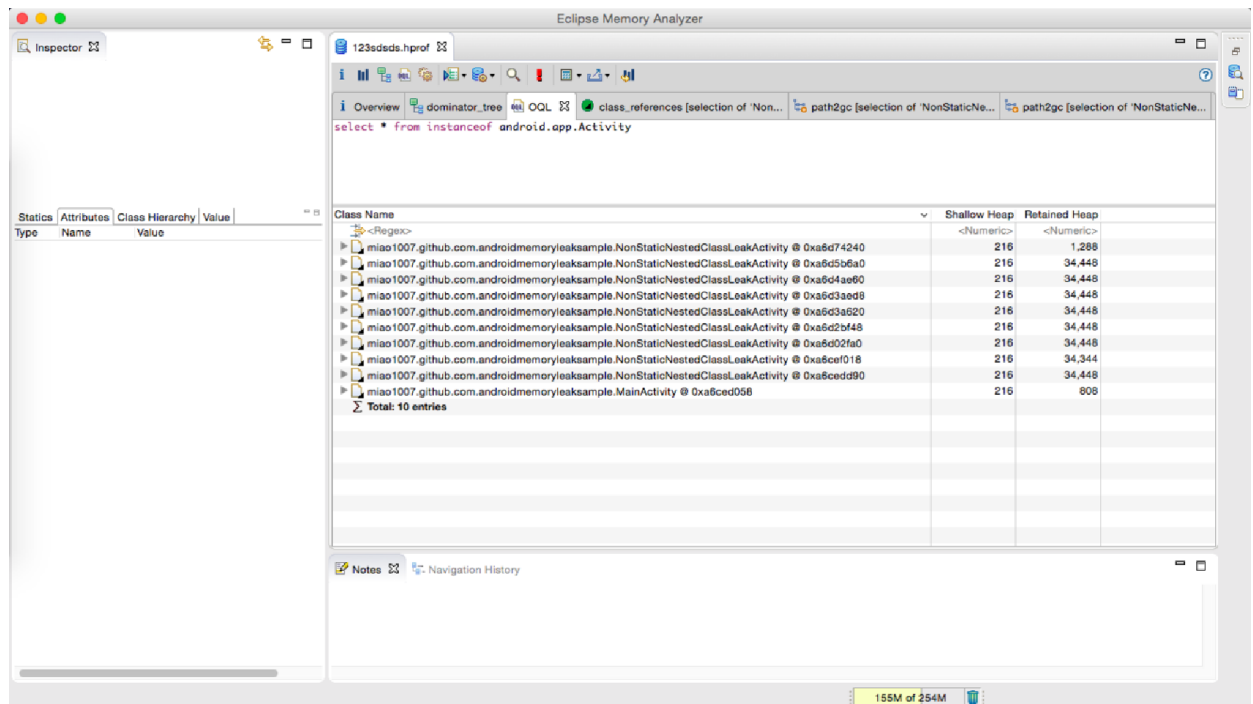
Android Studio dump Memory Analyze

- 点击左下角的Android
- 选中你的程序的包名
- 点击 initiates garbage collection on selected vm
- 点击 dump java heap for selected client

## 打开MAT，进行分析

MAT是对java heap中变量分析的一个工具，它可以用于分析内存泄露。

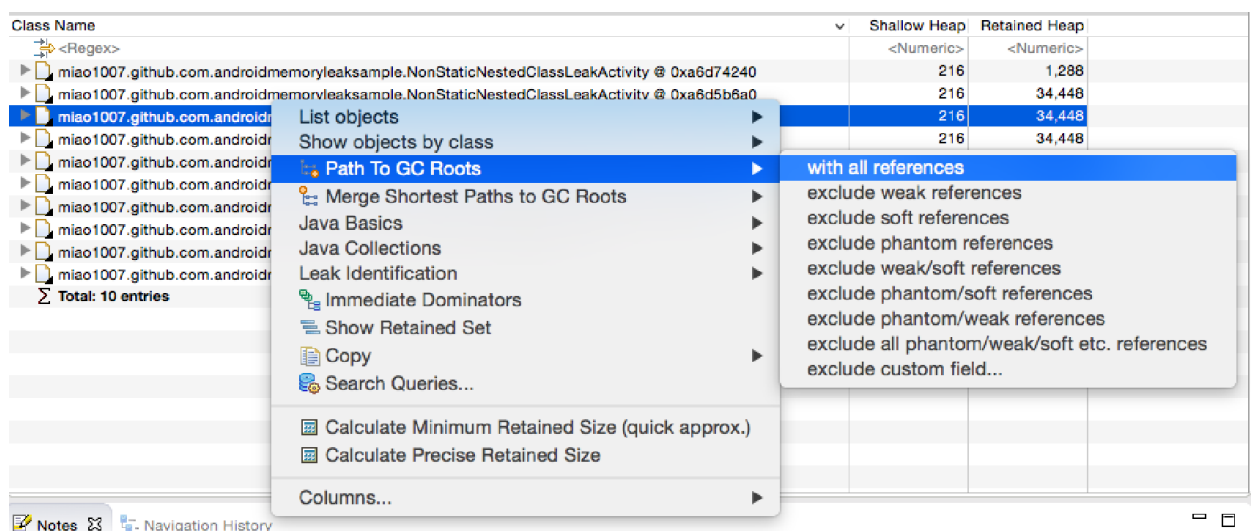
- 点击 OQL 图标
- 在窗口输入 `select * from instanceof android.app.Activity` 并按 `Ctrl + F5` 或者 `!` 按钮
- 奇迹出现了，现在你发现泄露了许多的activity
- 这个真是相当的不容乐观，我们来分析一下为什么GC没有回收它



EMA

在OQL ( Object Query Language ) 窗口下输入的查询命令可以获得所有在内存中的Activities，这段查询代码是不是非常简单高效呢？

点击一个activity对象，右键选中 Path to GC roots



GC root

Class Name	Shallow Heap	Retained Heap
<Regex>	<Numeric>	<Numeric>
miao1007.github.com.android.memoryleak.sample.NonStaticNestedClassLeakActivity @	216	34,448
this\$0 miao1007.github.com.android.memoryleak.sample.NonStaticNestedClassLeakActivity @	16	34,464
callback android.os.Message @ 0xa6d2b418	56	138,272
next android.os.Message @ 0xa6ce9fd8	56	172,816
next android.os.Message @ 0xa6d36350	56	207,360
next android.os.Message @ 0xa6cd3a8	56	241,904
next android.os.Message @ 0xa6cea0d8	56	276,344
<Java Local> java.lang.Thread @ 0xa624f4b0 main Thread	80	1,560
mMessages android.os.MessageQueue @ 0xa6ccb3c8	32	152
Σ Total: 2 entries		

Message in loop hold a reference to Activity

在打开的新窗口中，你可以发现，你的Activity是被 this\$0 所引用的，它实际上是匿名类对当前类的引用。this\$0 又被 callback 所引用，接着它又被 Message 中一串的 next 所引用，最后到主线程才结束。

任何情况下你在class中创建非静态内部类，内部类会（自动）拥有对当前类的一个强引用。

简  
(/)

一旦你把 Runnable 或者 Message 发送到 Handler 中，它就会被放入 LooperThread 的消息队列，并且被保持引用，直到 Message 被处理。发送postDelayed这样的消息，你输入延迟多少秒，它就会泄露至少多少秒。而发送没有延迟的消息的话，当队列中的消息过多时，也会造成一个临时的泄露。



(/apps)

**尝试使用static inner class来解决**

现在把 Runnable 变成静态的class

A

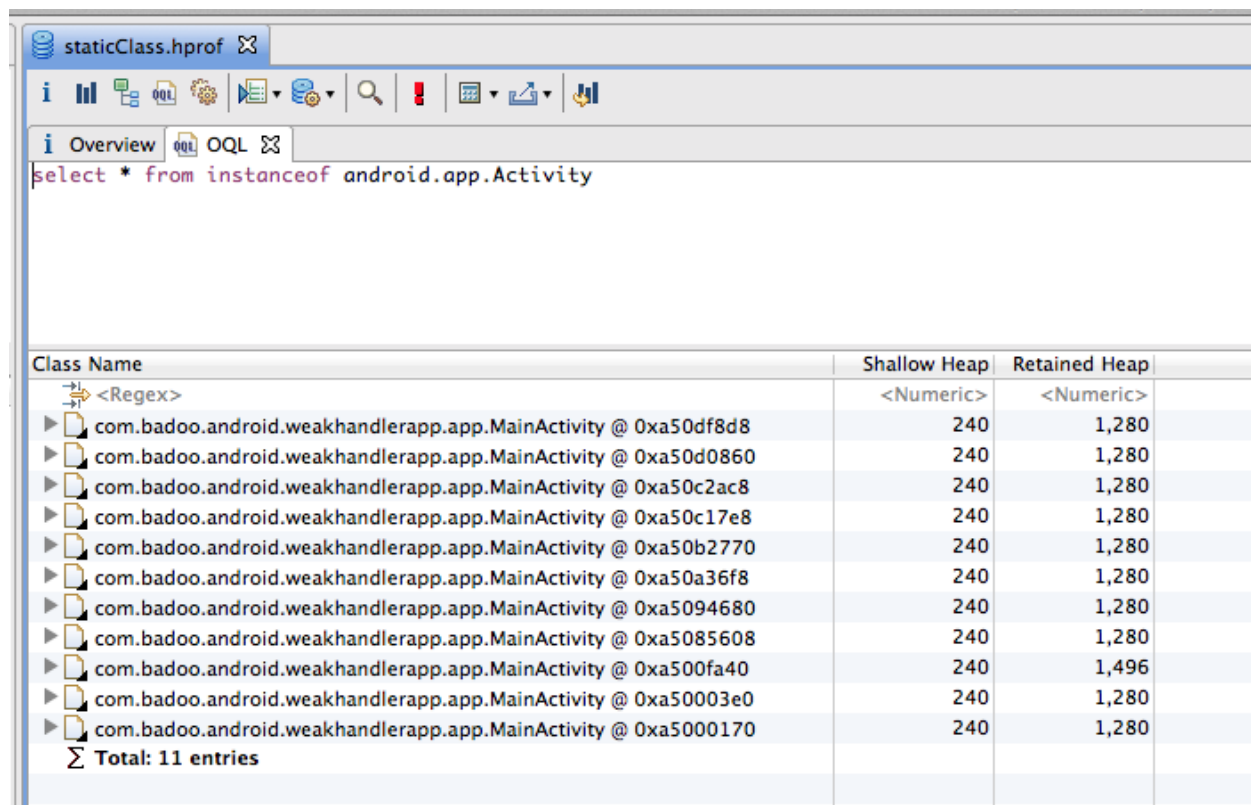


(/sign\_in)

```
9 public class MainActivity extends ActionBarActivity {
10
11     private Handler mHandler = new Handler();
12     private TextView mTextView;
13
14     @Override
15     protected void onCreate(Bundle savedInstanceState) {
16         super.onCreate(savedInstanceState);
17         setContentView(R.layout.activity_main);
18
19         mTextView = (TextView) findViewById(R.id.hello_text);
20         mHandler.postDelayed(new DoneRunnable(mTextView), 800000);
21     }
22
23     private static final class DoneRunnable implements Runnable {
24         private final TextView mTextView;
25
26         protected DoneRunnable(TextView textView) {
27             mTextView = textView;
28         }
29
30         @Override
31         public void run() {
32             mTextView.setText("Done");
33         }
34     }
35 }
```

### StaticClass

现在，摇一摇手机，导出内存

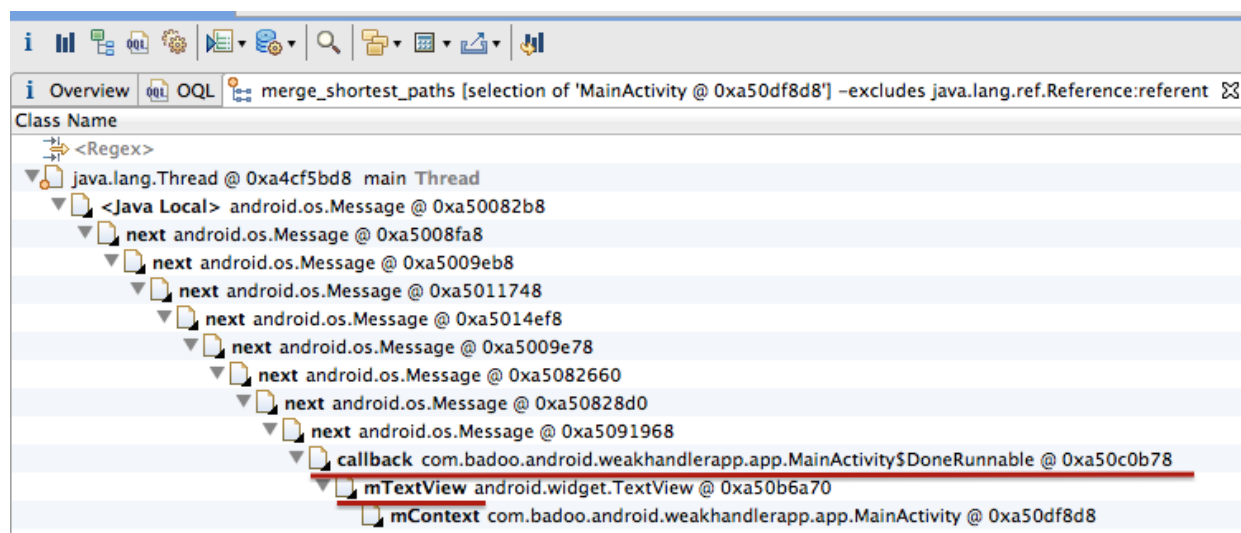


The screenshot shows the Android Studio interface with the memory profiler open. The 'Overview' tab is selected, displaying a table of memory allocations. The table has three columns: 'Class Name', 'Shallow Heap', and 'Retained Heap'. It lists 11 instances of 'com.badoo.android.weakhandlerapp.app.MainActivity', each with a unique memory address. The 'Shallow Heap' column shows a value of 240 for all instances, while the 'Retained Heap' column shows values ranging from 1,280 to 1,496. A 'Total: 11 entries' summary is at the bottom of the list.

Class Name	Shallow Heap	Retained Heap
<Regex>	<Numeric>	<Numeric>
com.badoo.android.weakhandlerapp.app.MainActivity @ 0xa50df8d8	240	1,280
com.badoo.android.weakhandlerapp.app.MainActivity @ 0xa50d0860	240	1,280
com.badoo.android.weakhandlerapp.app.MainActivity @ 0xa50c2ac8	240	1,280
com.badoo.android.weakhandlerapp.app.MainActivity @ 0xa50c17e8	240	1,280
com.badoo.android.weakhandlerapp.app.MainActivity @ 0xa50b2770	240	1,280
com.badoo.android.weakhandlerapp.app.MainActivity @ 0xa50a36f8	240	1,280
com.badoo.android.weakhandlerapp.app.MainActivity @ 0xa5094680	240	1,280
com.badoo.android.weakhandlerapp.app.MainActivity @ 0xa5085608	240	1,280
com.badoo.android.weakhandlerapp.app.MainActivity @ 0xa500fa40	240	1,496
com.badoo.android.weakhandlerapp.app.MainActivity @ 0xa50003e0	240	1,280
com.badoo.android.weakhandlerapp.app.MainActivity @ 0xa5000170	240	1,280
Total: 11 entries		

### StaticClass\_memory\_analyze

为什么又出现了泄露呢？我们看一看 Activities 的引用。



StaticClass\_memory\_analyze\_explained

看到下面的 mContext 的引用了吗，它被 mTextView 引用，这样说明，使用静态内部类还远远不够，我们仍然需要修改。

## 使用弱引用 + static Runnable

现在我们把刚刚内存泄露的罪魁祸首 – TextView改成弱引用。

```

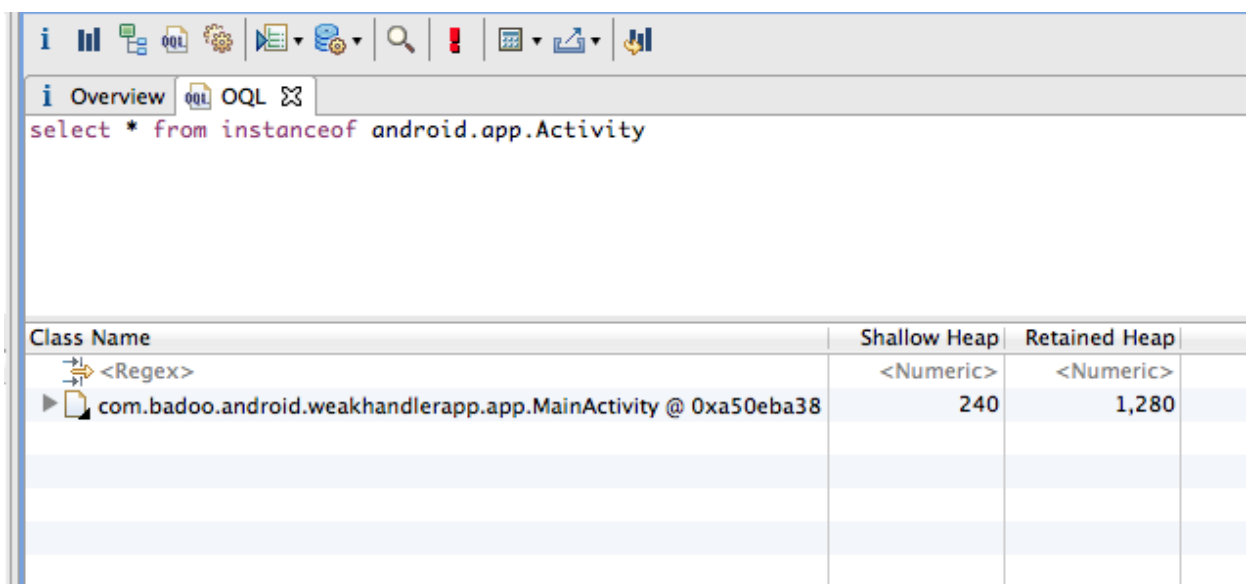
10
11 public class MainActivity extends ActionBarActivity {
12
13     private Handler mHandler = new Handler();
14     private TextView mTextView;
15
16     @Override
17     protected void onCreate(Bundle savedInstanceState) {
18         super.onCreate(savedInstanceState);
19         setContentView(R.layout.activity_main);
20
21         mTextView = (TextView) findViewById(R.id.hello_text);
22         mHandler.postDelayed(new DoneRunnable(mTextView), 800000);
23     }
24
25     private static final class DoneRunnable implements Runnable {
26         private final WeakReference<TextView> mTextViewRef;
27
28         protected DoneRunnable(TextView textView) {
29             mTextViewRef = new WeakReference<TextView>(textView);
30         }
31
32         @Override
33         public void run() {
34             final TextView textView = mTextViewRef.get();
35             if (textView != null) {
36                 textView.setText("Done");
37             }
38         }
39     }
40 }

```

StaticClassWithWeakRef\_code

再次注意我们对TextView保持的是**弱引用**，现在让它运行，摇晃手机

小心地操作WeakReferences，它们随时可以为空，在使用前要判断是否为空。



Class Name	Shallow Heap	Retained Heap
<Regex>	<Numeric>	<Numeric>
com.badoo.android.weakhandlerapp.app.MainActivity @ 0xa50eba38	240	1,280

StaticClassWithWeakRef\_memory\_analyze



哇！现在只有一个Activity的实例了，这回终于解决了我们的问题。

所以，我们应该记住：

- 使用静态内部类
- Handler/Runnable的依赖要使用弱引用。

如果你把现在的代码与开始的代码相比，你会发现它们大不相同，开始的代码易懂简介，你甚至可以脑补出运行结果。

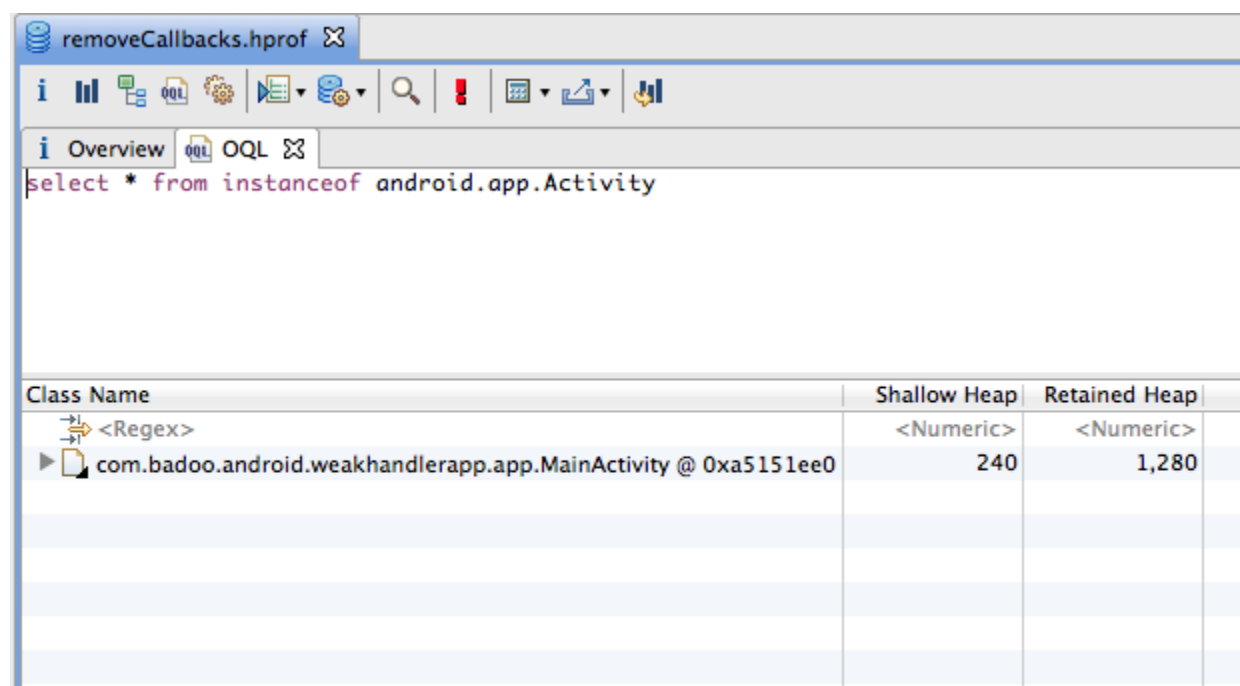
而现在的代码更加复杂，有很多的模板代码，当把 `postDelayed` 设置为一个短时间，比如 `50ms` 的情况下，写这么多代码就有点亏了。其实，还有一个更简单的方法。

## onDestroy中手动控制声明周期

Handler可以使用 `removeCallbacksAndMessages(null)`，它将移除这个Handler所拥有的 `Runnable` 与 `Message`。

```
//Fixed by manually control lifecycle
@Override protected void onDestroy() {
    super.onDestroy();
    myHandler.removeCallbacksAndMessages(null);
}
```

现在运行，旋转手机，导出内存



The screenshot shows the OQL (Object Query Language) tool in Android Studio. The query is `select * from instanceof android.app.Activity`. The results table shows the following data:

Class Name	Shallow Heap	Retained Heap
<Regex>	<Numeric>	<Numeric>
com.badoo.android.weakhandlerapp.app.MainActivity @ 0xa5151ee0	240	1,280

removeCallbacks\_memory\_analyze