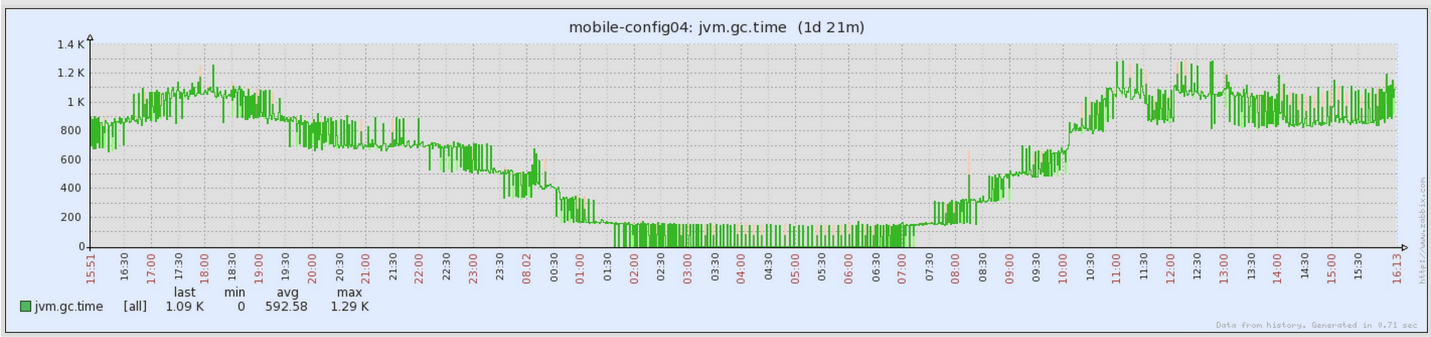
finalize导致Young GC 变慢

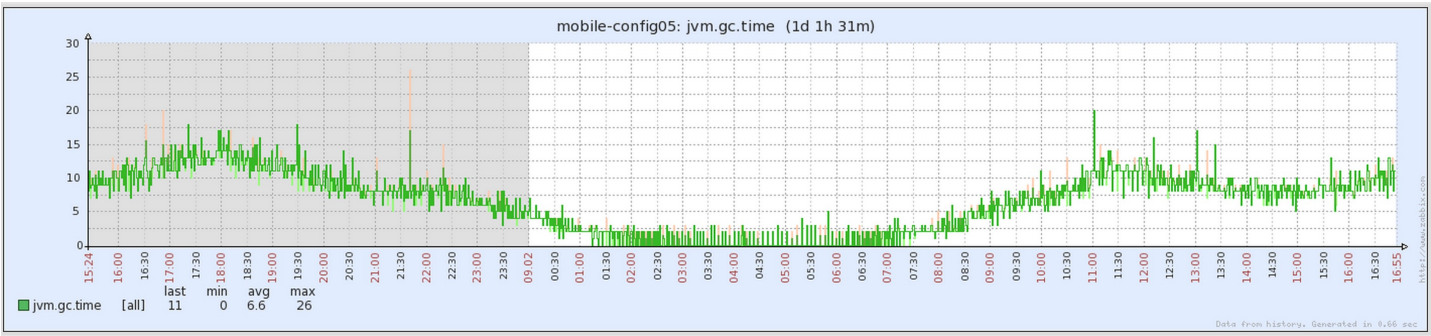
问题现象

SA同学反馈说线上两台机器的GC时间是其他机器的几百倍。

问题机器的GC时间：



正常机器的GC时间：



排查问题

step1. 执行jstat对比问题机器和正常机器，发现都只是在执行young gc问题机器的survivor区的使用量一直是60以上，而正常机器survivor区的使用量几乎接近于0。

由此现象分析，问题机器上存在某种对象，一次Young GC无法回收。所以需要把内存dump出来看一下。

step2. 问题机器上执行jmap，并sftp到Mac上，使用MAT分析如下：

heap

Overview | default_report org.eclipse.mat.api:suspects | dominator_tree

Class Name	Shallow Heap	Retained Heap	Percentage
<Regex>	<Numeric>	<Numeric>	<Numeric>
class java.lang.ref.Finalizer @ 0x7f8042768 System Class, Native Stack	16	66,841,480	78.91%
java.lang.ref.Finalizer @ 0x7ab0dbf98	40	66,841,448	78.91%
java.lang.ref.Finalizer @ 0x7ab0d1328	40	66,809,576	78.87%
org.restlet.engine.http.HttpRequest @ 0x7ab0d12a8	128	17,192	0.02%
org.eclipse.jetty.server.AsyncHttpConnection @ 0x7aaab9e28	120	3,280	0.00%
org.restlet.engine.http.HttpResponse @ 0x7ab0dbf28	112	3,088	0.00%
org.eclipse.jetty.server.Request @ 0x7aaaba368	184	3,032	0.00%
org.restlet.data.Form @ 0x7ab0d1550	16	2,824	0.00%
org.restlet.ext.servlet.internal.ServletCall @ 0x7ab0d1178	96	912	0.00%
org.eclipse.jetty.io.nio.SelectChannelEndPoint @ 0x7aaab9d08	80	800	0.00%
org.eclipse.jetty.http.HttpParser @ 0x7aaab9fc0	96	280	0.00%
java.lang.String @ 0x7ab0d29c0 com.meituan.imeituan/611 (unknown, iPhone	32	200	0.00%
java.lang.String @ 0x7ab0d2040 api.mobile.meituan.com	32	96	0.00%
org.eclipse.jetty.server.Response @ 0x7aaaba460	56	56	0.00%
org.eclipse.jetty.server.HttpInput @ 0x7ab0e3c08	24	24	0.00%
org.restlet.data.Parameter @ 0x7ab1045b8	24	24	0.00%
org.restlet.data.Parameter @ 0x7ab1045d0	24	24	0.00%
class javax.servlet.http.Cookie[] @ 0x7fa2d07f0	0	0	0.00%
Total: 16 entries			
java.lang.Object @ 0x7b89c9678	16	16	0.00%
Total: 2 entries			
com.sun.tools.javac.zip.ZipFileIndex @ 0x7b9ac2e70	72	2,159,544	2.55%
org.eclipse.jetty.webapp.WebAppClassLoader @ 0x7b86f8f60	88	2,091,904	2.47%
org.springframework.beans.factory.support.DefaultListableBeanFactory @ 0x7b8d580	200	1,086,528	1.28%
com.sun.tools.javac.zip.ZipFileIndex @ 0x7b9c2f620	72	668,072	0.79%

从MAT展现的结果可以发现，大量的Finalizer对象占用了79%的堆内存。猜测可能跟Finalizer的实现机制有关。

step4. google资料，查看Finalizer的机制。重写了Object对象的finalize方法的类对象，GC的方式和普通对象有很大的差异。对于实现了finalize方法的类，在创建该类的对象时，也会创建一个Finalizer对象，Finalizer的referent引用指向此对象。在发生GC时，实现finalize的对象不会被直接回收，而是会被JVM识别出，JVM会将Finalizer对象放入Finalizer的static的Queue中。所以，一次Young GC不会回收掉实现了finalize方法的对象，而是转移到survivor区。JVM在加载Finalizer对象的时候，会启动一个名为“Finalizer”的线程，负责清理Finalizer的Queue中的对象。清理的过程，就是执行referent引用对象的finalize方法，并同时从队列中移除。被清理之后，再没有引用指向实现了finalize方法对象了，那么在下一次GC的时候，此对象就可以被回收了。

根据以上说明，Survivor区始终保持60以上的使用率就是Finalizer的机制导致的，通过MAT查看Finalizer 的referent引用的对象是org.restlet.engine.http.HttpResponse和org.restlet.engine.http.HttpRequest。

step5. 在IDE中查找restlet那两个类，没有发现其实现了finalize方法，觉得很诡异，暂时搁置问题。

第二天意识到这是典型的类加载冲突导致的。登录线上机器，查看类加载情况，问题机器的类加载情况：

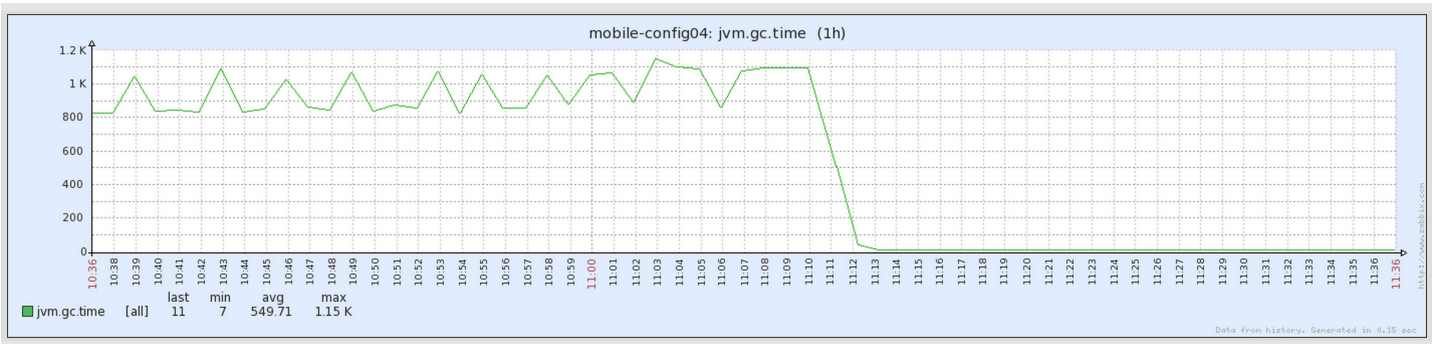
```
org.restlet.engine.http.HttpResponse -> /opt/meituan/mobile/config/webroot/WEB-INF/lib/org.
```

正常机器的类加载情况：

```
org.restlet.engine.http.HttpResponse -> /opt/meituan/mobile/config/webroot/WEB-INF/lib/org.
```

类加载冲突的问题得到验证。Intellij Idea只导入了org.restlet-2.0.10.jar，导致我全文查找“finalize”没有找到。重新导入org.restlet-2.0.0.jar，发现其HttpRequest和HttpRespose确实重写了finalize方法。

step6. exclude掉冲突的jar，重新发布后，gc time的改善情况如下：



gc恢复正常。

结论

问题解决了，但是为何重写了finalize方法的对象回收效率这么低呢？下面一句话引自Effective Java Edition 2中Joshua Bloch的一句话：

```
"Oh, and one more thing: there is a severe performance penalty for using finalizers. On my

Adding a finalizer increases the time to 2,400 ns. In other words, it is about 430 times sl
```

所以，要谨慎重写finalize方法，对于重写了finalize方法的第三方包也要小心使用。

另外，java的类冲突问题会引发各种莫名其妙的问题，在发布之前的jar冲突检测，可以有效的规避此类问题。

参考资料

http://www.fasterj.com/articles/finalizer1.shtml ----Finalizer对象的生命周期