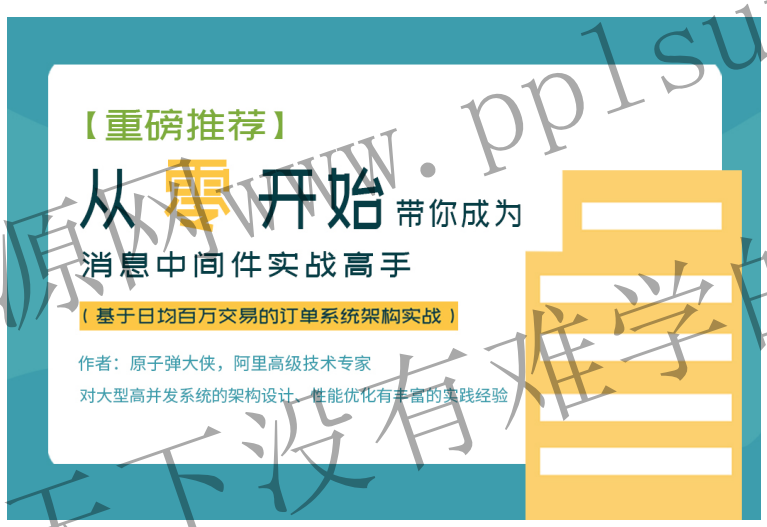
图文 084、动手实验：JVM栈内存溢出的时候，应该如何解决?

详情 评论

动手实验：

JVM栈内存溢出的时候，应该如何解决?

狸猫技术窝专栏上新，基于**真实订单系统**的消息中间件（mq）实战，重磅推荐：



未来3个月，我的好朋友原子弹大侠将带你一起，全程实战，360度死磕MQ

（点击下方蓝字进行试听）

从 0 开始带你成为消息中间件实战高手

重要说明：

如何提问：每篇文章都有评论区，大家可以尽情在评论区留言提问，我都会逐一答疑

（ps：评论区还精选了一些小伙伴对**专栏每日思考题的作答**，有的答案真的非常好！大家可以通过看别人的思路，启发一下自己，从而加深理解**）**

如何加群：购买了狸猫技术窝专栏的小伙伴都可以加入**狸猫技术交流群。**

（群里有不少**一二线互联网大厂的助教**，大家可以一起讨论交流各种技术）

具体**加群方式**请参见文末。

（**注**：以前通过其他专栏加过群的同学就不要重复加了）

1、前文回顾

之前的文章，我们分析了Metaspace区域是如何内存溢出的，同时还带着大家分析了一下内存快照。

今天这篇文章，我们就带大家分析一下JVM栈内存溢出的时候，怎么来解决。

2、栈内存溢出能依托之前的办法解决吗?

首先大家思考一个问题：栈内存溢出能按照之前的方法解决吗?

也就是说，GC日志、内存快照，这些东西对解决栈内存溢出有帮助吗?

首先明确一点，栈内存溢出跟堆内存是没有关系的，因为他的本质是一个线程的栈中压入了过多方法调用的栈桢，比如几千次方法调用的几千个栈桢。

此时就导致线程的栈内存不足，无法放入更多栈桢了。

所以GC日志对你有用吗?

没用!因为GC日志主要是分析堆内存和Metaspace区域的一些GC情况的，就线程的栈内存和栈桢而言，他们不存在所谓的GC。

如果大家还记得之前我们画的图，就应该知道，调用一个方法时在栈里压入栈桢，接着执行完整个方法，栈桢从栈里出来，然后一个线程运行完毕时，他的栈内存就没了。

所以本身这块内存不存在所谓的GC和回收，调用方法就给栈桢分配内存，执行完方法就回收掉那个栈桢的内存。

那么内存快照呢?

内存快照主要是分析一些内存占用的，同样是针对堆内存和Metaspace的，所以对线程的栈内存而言，也不需要借助这个东西。

3、示例代码

```
public class Demo2 {

  public static long counter = 0;

  public static void main(String[] args) {
    work();
  }

  public static void work() {
    System.out.println("目前是第" + (++counter) + "次调用方法");
    work();
  }

}
```

使用的JVM参数如下:

```
-XX:ThreadStackSize=1m
-XX:+PrintGCDetails
-Xloggc:gc.log
-XX:+HeapDumpOnOutOfMemoryError
-XX:HeapDumpPath=./
-XX:+UseParNewGC
-XX:+UseConcMarkSweepGC
```

4、运行代码后分析异常报错信息的调用栈

接着我们运行代码让他产生栈内存溢出的报错，如下：

at com.limao.demo.jvm.Demo2.work(Demo2.java:13)
at com.limao.demo.jvm.Demo2.work(Demo2.java:13)
at com.limao.demo.jvm.Demo2.work(Demo2.java:13)
at com.limao.demo.jvm.Demo2.work(Demo2.java:13)
at com.limao.demo.jvm.Demo2.work(Demo2.java:13)
at com.limao.demo.jvm.Demo2.work(Demo2.java:13)
at com.limao.demo.jvm.Demo2.work(Demo2.java:13)
at com.limao.demo.jvm.Demo2.work(Demo2.java:13)
at com.limao.demo.jvm.Demo2.work(Demo2.java:13)

实际上我们会在这里看到大段大段的如上所示的异常，也就是说，他会直接告诉你这个栈内存溢出的问题，是因为你拼命的调用Demo2这个类的work()方法时发生的。

因此就栈内存溢出而言，我们定位和解决问题非常的简单，你只要把所有的异常都写入本地日志文件，那么当你发现系统崩溃了，第一步就去日志里定位一下异常信息就知道了。

比如，昨天我们通过异常信息直接定位出来是Metaspace区域出的异常，然后分析一下GC日志就完全知道发生溢出的全过程，接着再分析一下MAT的内存快照，就知道是哪个类太多导致的异常。

今天的栈内存溢出，我们直接去日志文件里看到是栈内存溢出：Exception in thread "main" java.lang.StackOverflowError。

此时心里就有数了，然后直接看一下对应报错的方法就可以了。知道是哪个方法，直接去代码中定位问题即可。

5、本文总结

今天的文章带着大家学习了一下栈内存溢出的解决办法，其实这是最容易定位和解决的一种异常了，大家只要有一个好习惯，就是把异常信息都写入本地日志文件，系统崩溃了直接看日志就知道怎么回事了。

End
