# 前言

模拟一个死锁、高cpu、高内存的demo，然后通过arthas分析。

# 总览

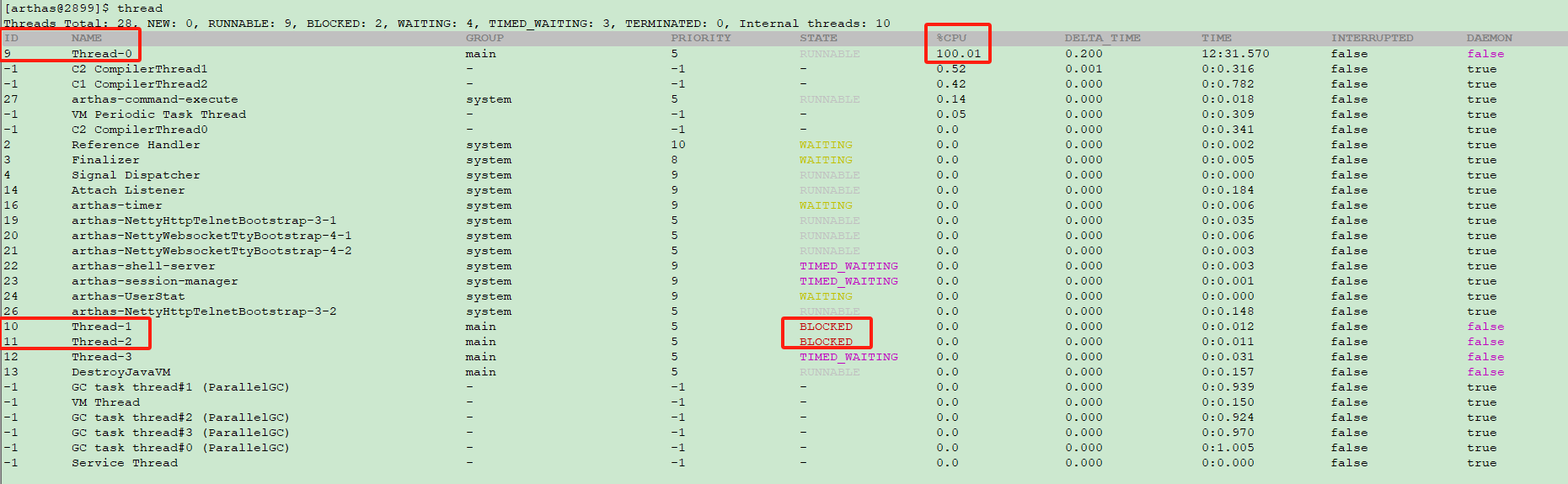
## dashboard

可以通过dashboard查看整个程序概况，包含线程、内存、GC、运行环境信息。



备注：线程id是arthas自己定义的，跟top的没关系。

## thread

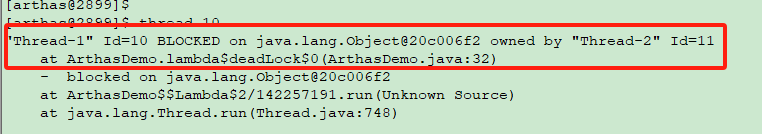


9号线程cpu占了100%，10、11号线程死锁

# 检查死锁

## thread n

通过thread 线程id查看线程堆栈。

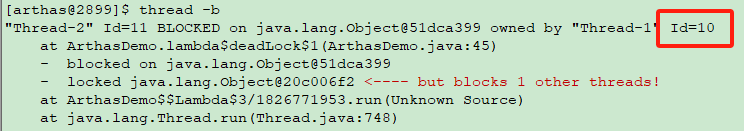


线程10的对象Object@20c006f2阻塞了，锁住该对象的是线程11。

锁代码在ArthasDemo.java:32，也就是ArthasDemo#deadLock内(通过反编译是41行)

## thread -b

查看死锁线程

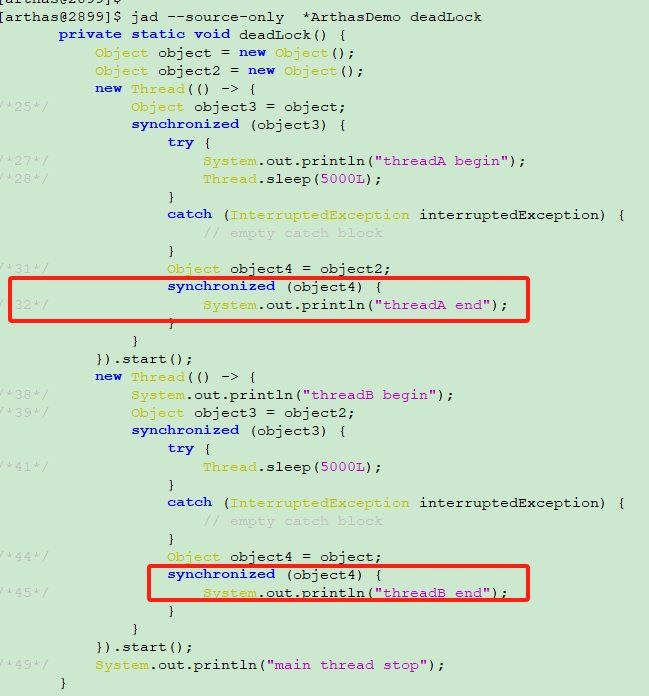


线程11的对象Object@51dca399阻塞了，锁住该对象的是线程10。

锁代码在ArthasDemo.java:45，也就是ArthasDemo#deadLock内(通过反编译是44行)

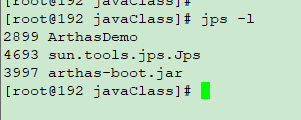
## Jad反编译

jad --source-only \*.ArthasDemo deadLock



## JDK自带方式

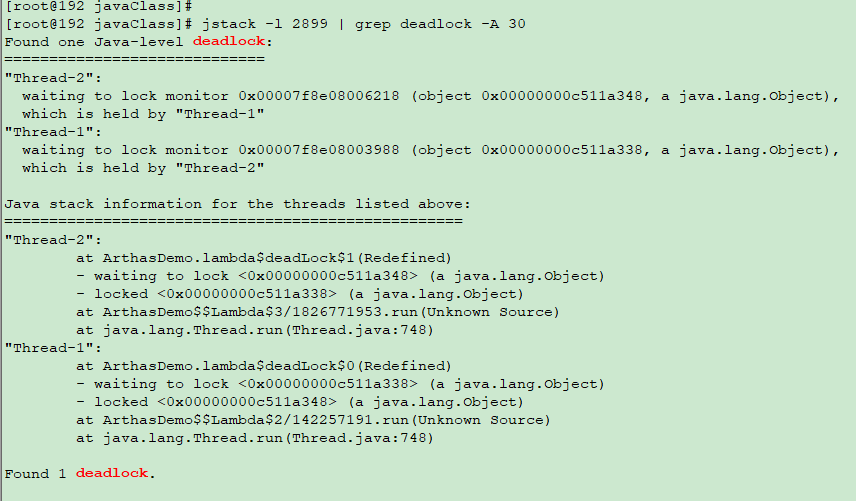
### jps -l 查看pid



### **jstack查看死锁**

**jstack -l pid | grep deadlock -A 30**

备注： -A表示查看关键字deadlock 后面的第N行。



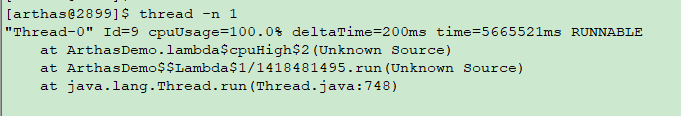
### **jstack弊端**

**能看锁死，但有可能看不到锁在代码第几行。**

# 检查cpu高占比

## thread -n

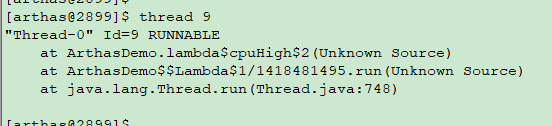
thread -n M 查看topM条最耗CPU的线程。



等知是线程id 9最耗cpu(其实dashboard也看到)

## thread n

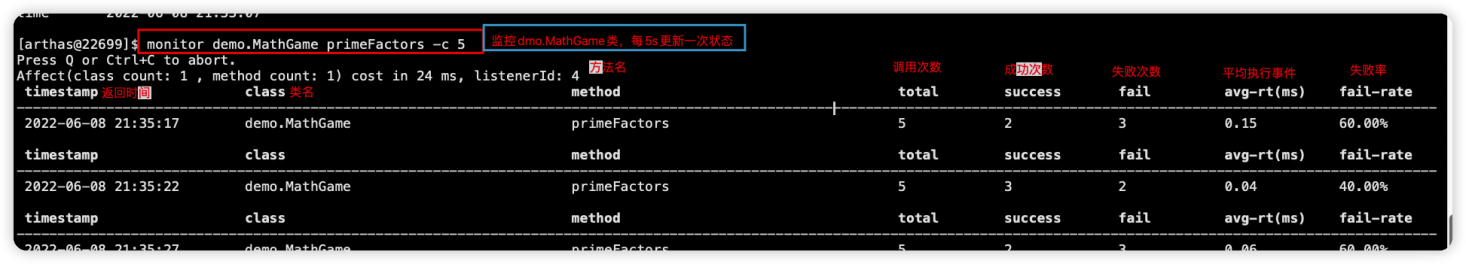
查看具体线程堆栈



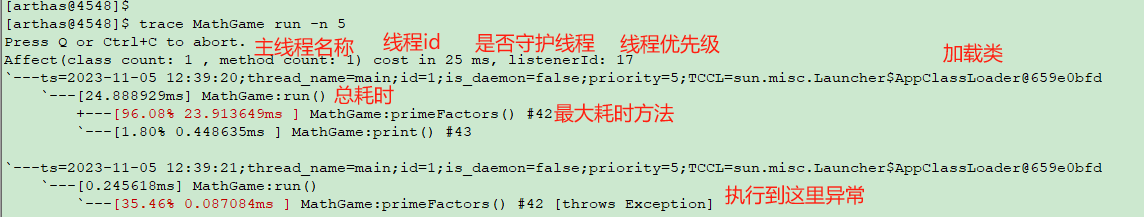
## 监控应用

实际生产中，往往调用完线程就会消亡，需要监控。

monitor ArthasDemo cpuHigh -c 5 --每5s更新监控一次方法cpuHigh



trace ArthasDemo cpuHigh -n 5 '#cost > 1' (执行五次并筛选大于1ms的调用)



stack MathGame primeFactors -n 5 '#cost > .5' 查看调用链路(跟好像功能跟trace一样)

