Netty

项目实战

**Netty项目实战优化文档**

**版本1.0**

**2019年12月**

**文档修改历史**

|  |  |  |  |
| --- | --- | --- | --- |
| **版本号** | **日期** | **修改人** | **修订内容** |
| 1.0 | 2019年12月12日 | 董杨炀 | Netty项目实战 |

**目录 Content**

[综述 3](#_Toc27475383)

[范围 3](#_Toc27475384)

[读者 3](#_Toc27475385)

[术语 3](#_Toc27475386)

[前言 3](#_Toc27475387)

[开发实战 4](#_Toc27475388)

[服务端 4](#_Toc27475389)

[客户端 4](#_Toc27475390)

[服务端优化说明 4](#_Toc27475391)

[客户端优化说明 11](#_Toc27475392)

[测试实战 11](#_Toc27475393)

[部署监控 11](#_Toc27475394)

[CPU瓶颈 12](#_Toc27475395)

[内存瓶颈 13](#_Toc27475396)

[代码性能 14](#_Toc27475397)

[运维实战 14](#_Toc27475398)

[Linux内核调优 14](#_Toc27475399)

[附录 15](#_Toc27475400)

[808网关半裸奔参考 15](#_Toc27475401)

[Netty全裸奔参考 16](#_Toc27475402)

# 综述

## 范围

本文档主要描述Netty项目实战优化文档

## 读者

本文档面向的读者为项目经理、系统设计人员、各模块组开发人员、模块测试人员。

## 术语

|  |  |  |
| --- | --- | --- |
| **序号** | **术语** | **说明** |
| 1 | Netty | Netty 是一个基于NIO的客户、服务器端的编程框架 |

## 前言

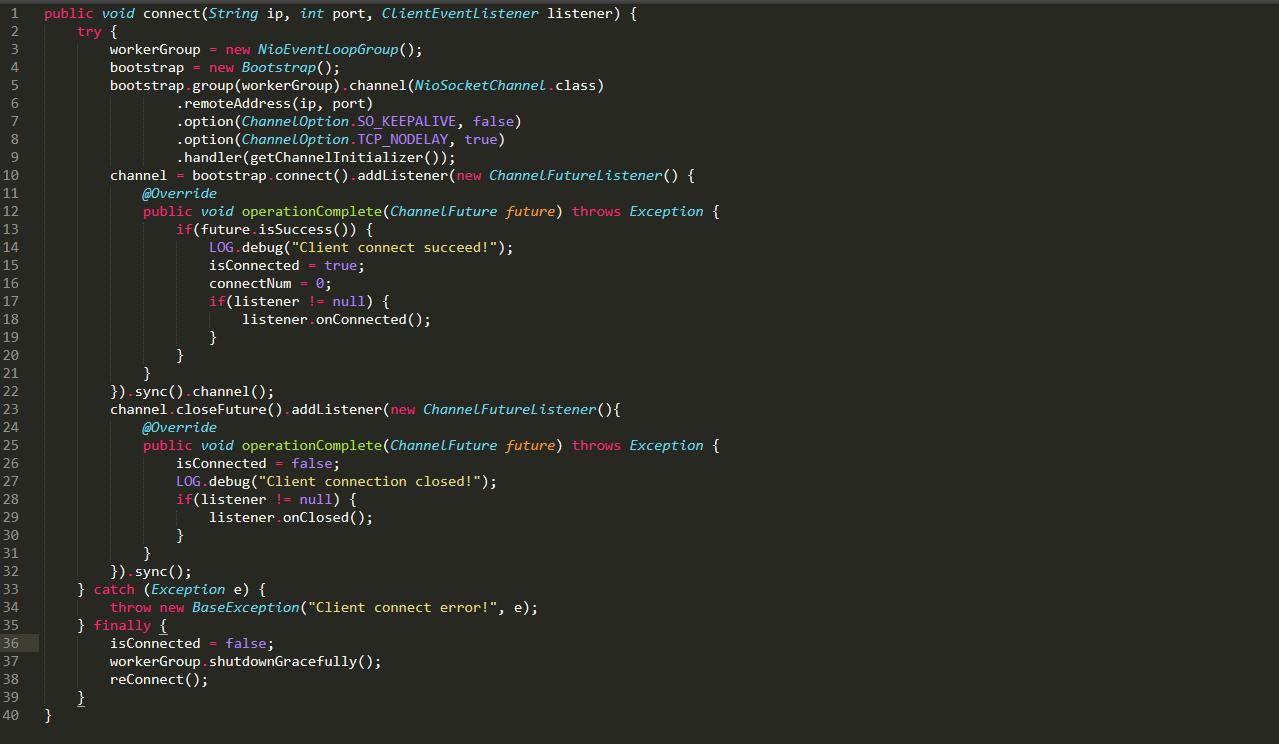
该文档对于Netty基本组件描述忽略！开发人员自行学习！这里只体现项目实战中的优化点。

# 开发实战

## 服务端



## 客户端



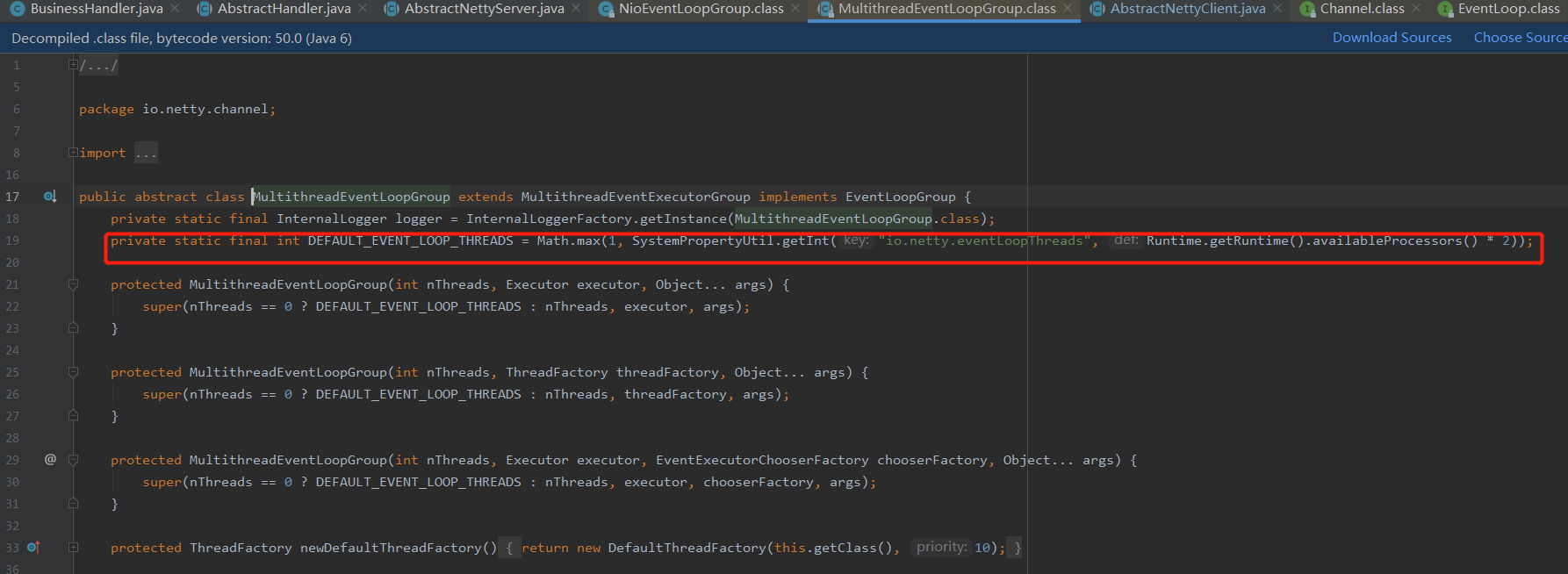
## 服务端优化说明

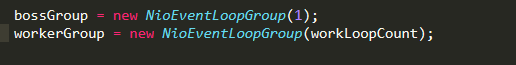
#### ChannelOption

|  |  |  |
| --- | --- | --- |
| 参数名 | 建议值 | 描述 |
| SO\_BACKLOG | 4096 | Backlog主要是指当ServerSocket还没执行accept时，这个时候的请求会放在os层面的一个队列里，这个队列的大小即为backlog值。  这个参数对于大量连接涌入的场景非常重要，例如服务端重启，所有客户端自动重连，瞬间就会涌入很多连接，  如backlog不够大的话，可能会造成客户端接到连接失败的状况，再次重连，结果就会导致服务端一直处理不过来  backlog的默认值为os对应的net.core.somaxconn，调整backlog队列的大小一定要确认ulimit -n中允许打开的文件数是够的。 |
| CONNECT\_TIMEOUT\_MILLIS | 5000 | 建立连接超时时间 |
| ALLOCATOR | PooledByteBufAllocator.DEFAULT | 开启Netty默认的内存池 |
| SO\_KEEPALIVE | Boolean.FALSE | 禁用Linux内核的TCP心跳自实现userEventTriggered |
| TCP\_NODELAY | Boolean.TRUE | 禁用Nagle算法, 对通信场景实时要求高需要设置为TRUE |

#### EventLoopGroup

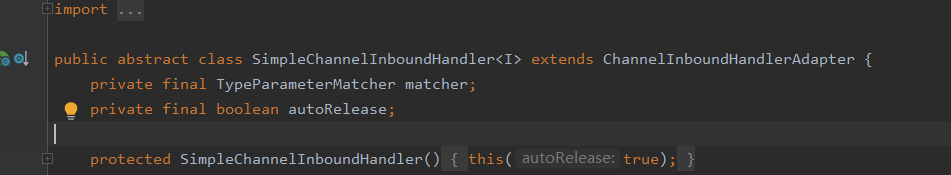
需手动指定bossGroup与workGroup的线程数。在docker中，Netty无法获取当前宿主主机的CPU可用核数，将会默认值1\*2。建议根据部署情况进行配置核数。建议workGroup线程数为CPU核数\*2，bossGroup线程数为1。

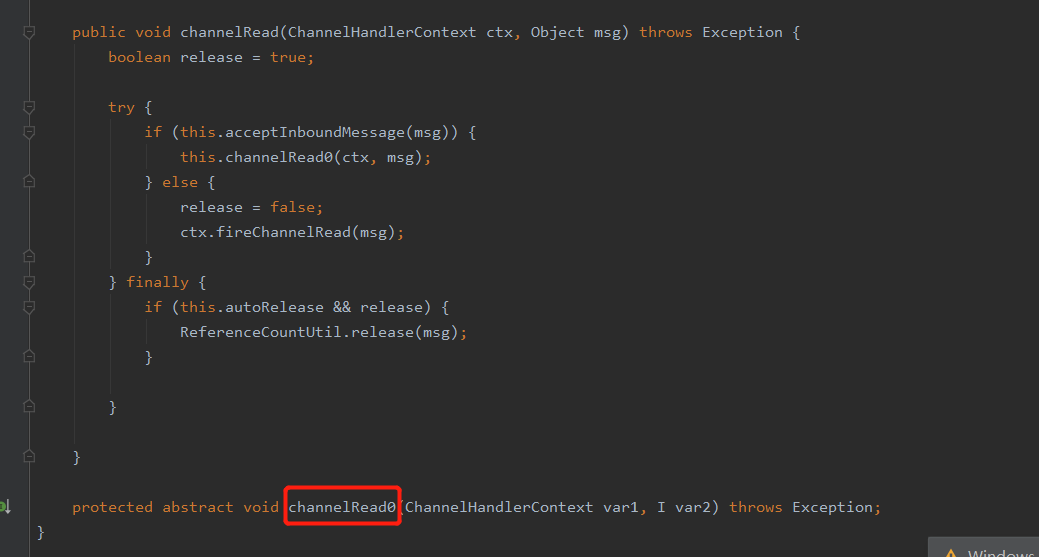




#### ChannelInboundHandlerAdapter

建议使用SimpleChannelInboundHandler。Netty帮助释放ByteBuf,如果你要自实现，请一定要释放ByteBuf，否则将会有内存溢出的风险！





#### encode/decode

开发应严格按照ByteBuf进行编码解码。避免使用Java原生的ByteBuffer!

在开启内存池后,decode所接收的ByteBuf均为Netty池化实现。

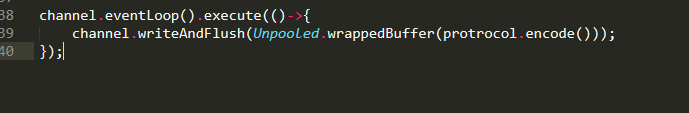
开发所能控制是encode,应尽可能的使用池化的ByteBuf！

使用池化的ByteBuf主要是为了减少内存GC的频率。当然一般Netty应用内存不是瓶颈！内存瓶颈标志为应用频繁的触发FGC，FGC将导致CPU阻塞，系统卡顿，FGC频率一但过高，应用离内存溢出不远矣！

#### writeAndFlush

在写writeAndFlush时应尽可能用Netty的EventLoop来调度。

正确的使用姿势为:



错误使用姿势为:



#### userEventTriggered

心跳空闲检查,用来扫描清理空闲连接。释放端口资源。实际设备都会有固定频率心跳。例如GB32960，建议Tbox到TSP的心跳频率为4秒。TSP空闲检查为8秒。满足Tbox进行10秒补发的需求。(建议空闲检查<=60秒)

其中运维应该优化Linux内核的net.ipv4.tcp\_fin\_timeout，降低CLOSE\_WAIT持续时间。

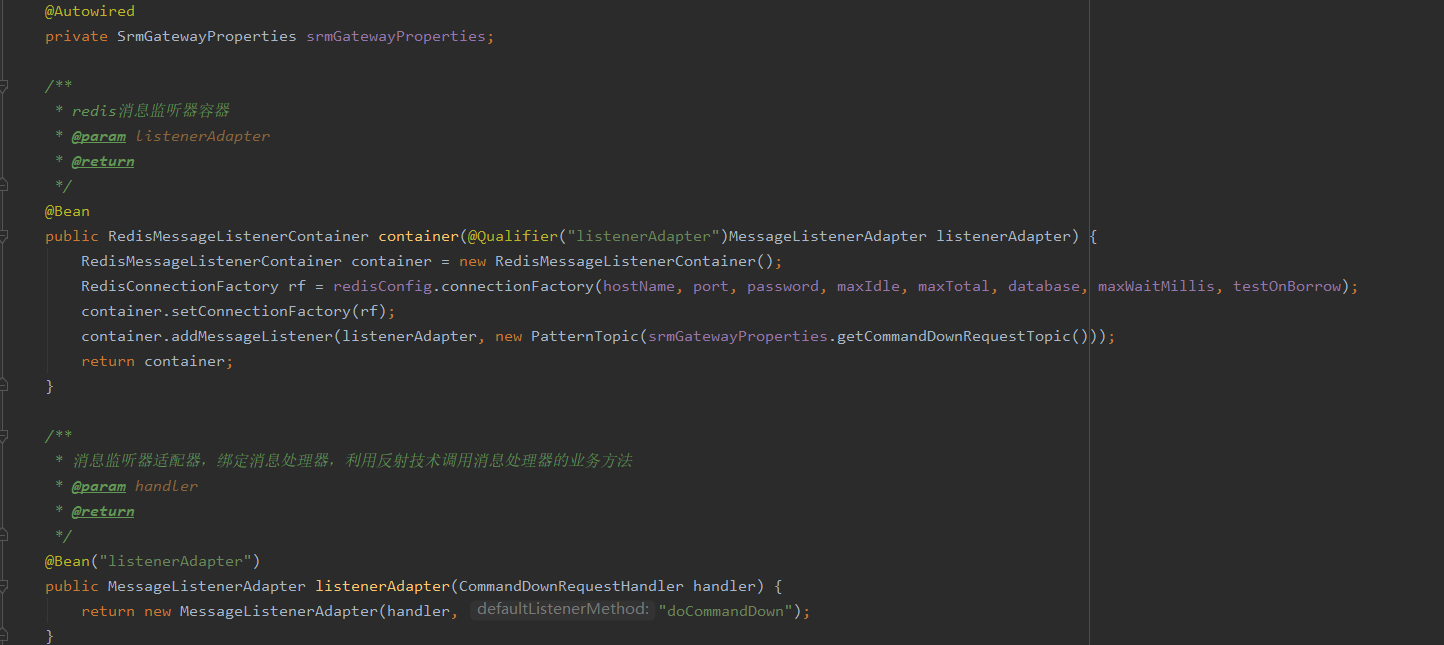
#### 代码设计优化

高性能网关设计一般都采用Redis+kafka作为解耦中间件！项目中难免使用到缓存与消息队列。线性逻辑耗时风险操作都应异步非阻塞，防止Netty的eventLoopGroup阻塞。

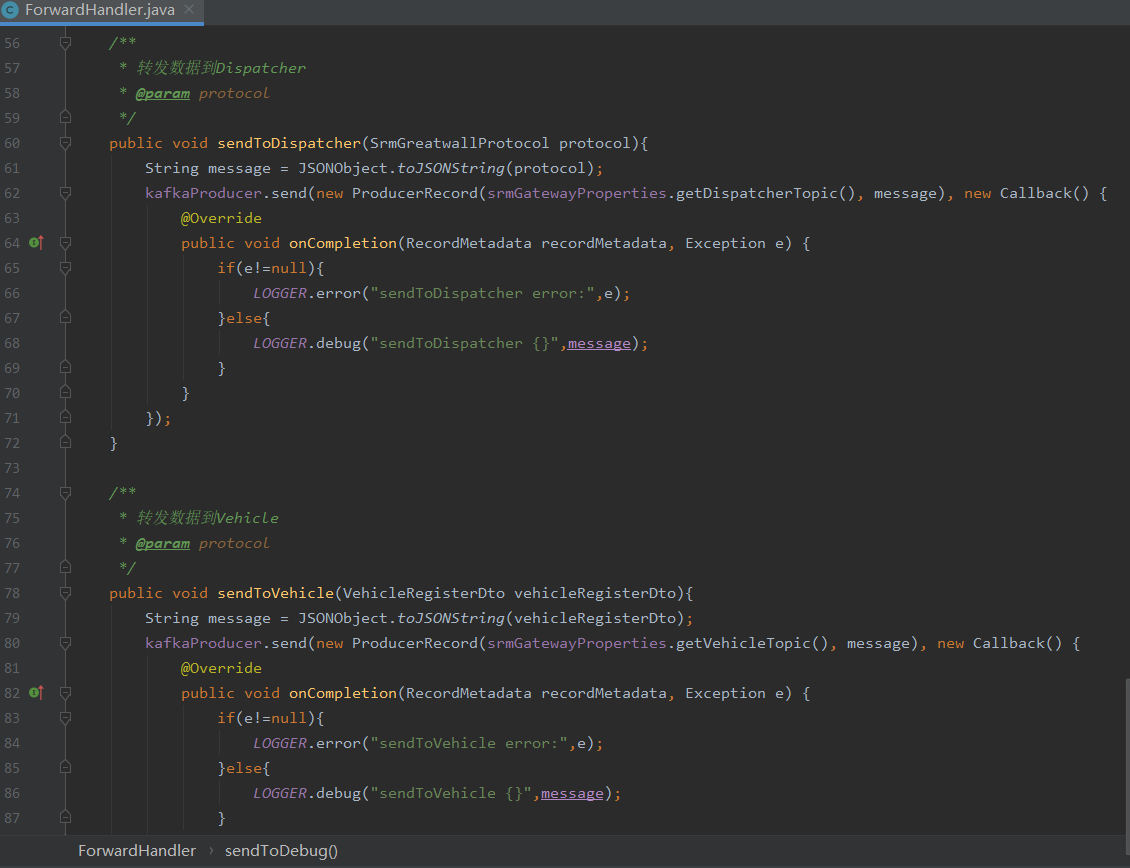
1. Redis缓存应该与内存结合使用实现二级缓存。(注意及时清理内存)



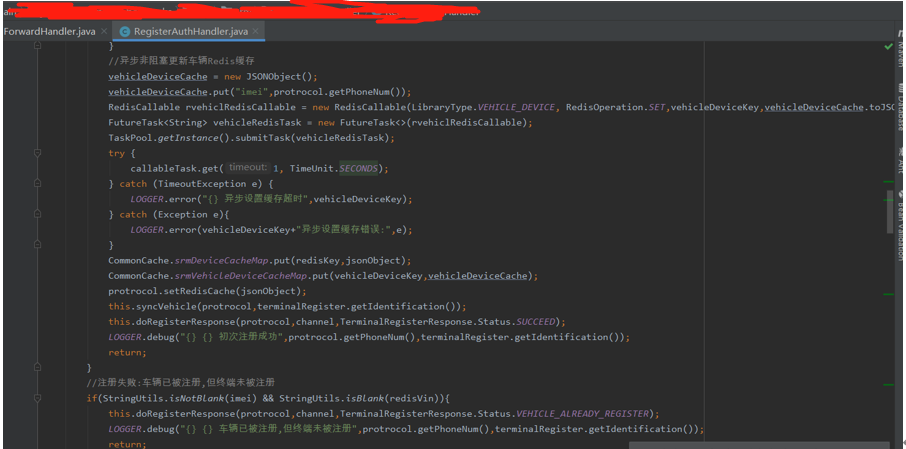
1. 网关对接终端的远控指令下发应用Redis的订阅发布机制。用kafka需要配置不同的消费者groupId,在pods部署集群时不太友好。



1. Kafka本身为异步处理，只需常规优化配置即可



1. 所有的Redis操作应进行异步非阻塞实现。避免Redis宕机或者网络波动时导致网关出现问题。



## 客户端优化说明

#### 重发策略

上行无响应或响应延迟时必须以固定频率重发最大次数后方可断开原连接,再进行重连策略

#### 重连策略

重连间隔必须以2的N次方秒递增,当达到某个固定值，以固定频率重连。

# 测试实战

## 部署监控

准备数据略,建议购买Xmeter压测。这里用16c主机压测808网关阐述。

直接java -jar运行。打开jvisualvm监控端口。设置G1回收。JVM内存8G

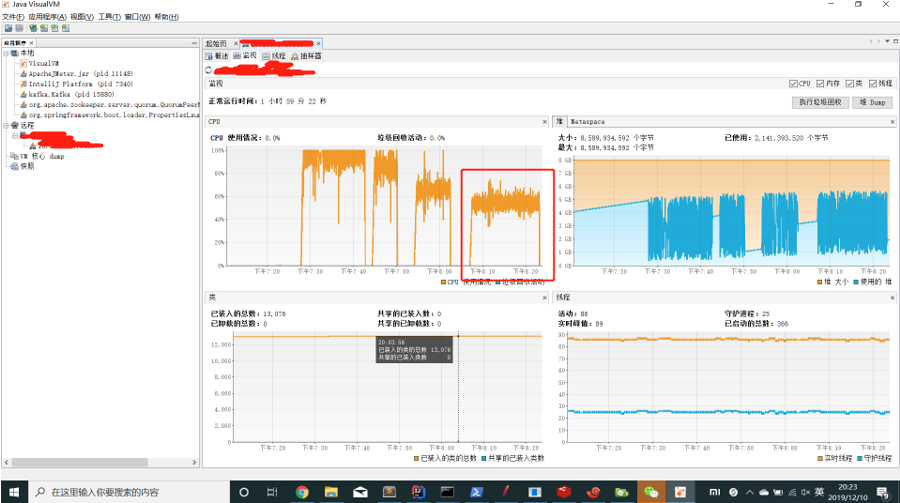


压测频率1秒1条报文。

以下压测结果为:Linux内核未调优的结果

如下为如何寻找应用CPU瓶颈与内存瓶颈实操！

## CPU瓶颈



50000车，40000车，30000车，25000车分别递减压测观察CPU变化分析所得。当25000车，CPU稳定在60%-80%。由此可见，25000车时应用运行的最佳状态，此时TSP稳定在2.4W/S。压测频率1秒1条。实车10秒1条。基本16C单节点可支撑30万车辆左右。单节点句柄至少设置在100万。



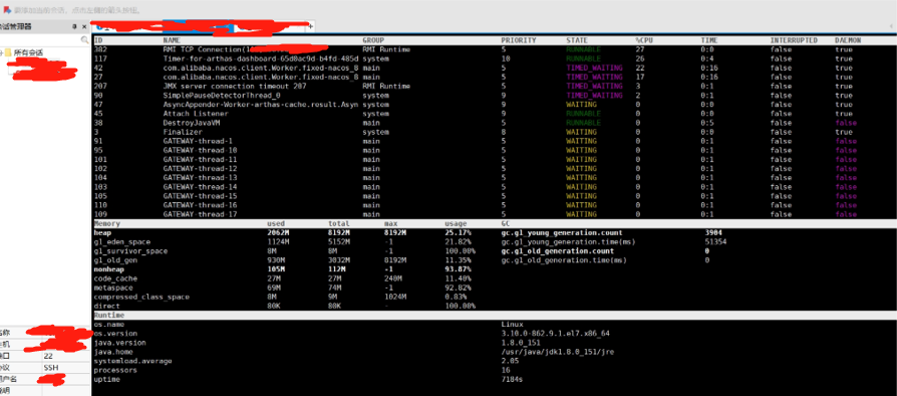






## 内存瓶颈

50000车，40000车，30000车，25000车分别递减压测观察内存变化与Arthas内存GC监控综合分析所得。内存并不是网关的瓶颈，如果部署在16c32g主机。内存建议设置24G。网关应定制化部署，尽可能单节点发挥宿主主机的最佳性能！



## 代码性能

Arthas监控业务处理方法执行100000次。并没有执行超过10ms以上的方法。

# 运维实战

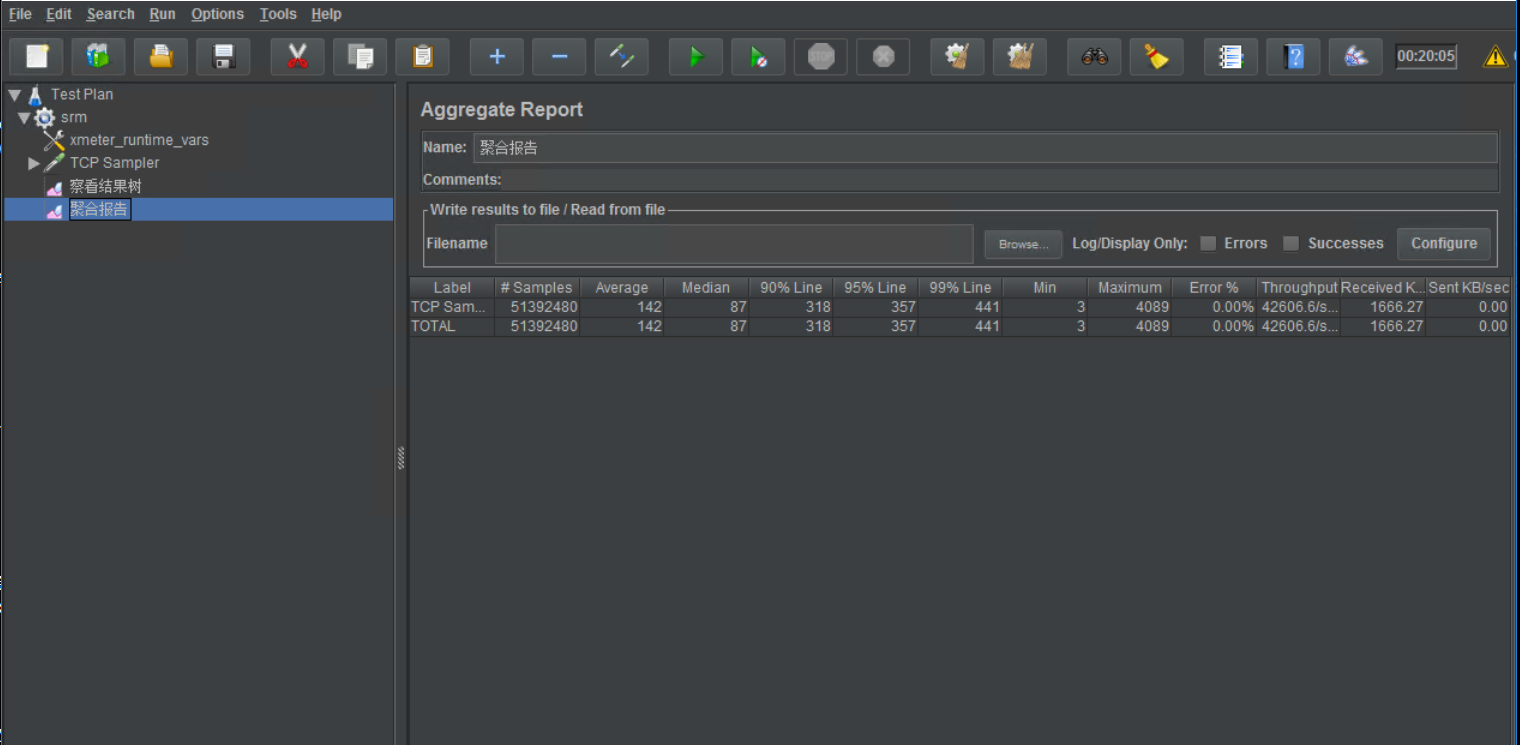
## Linux内核调优

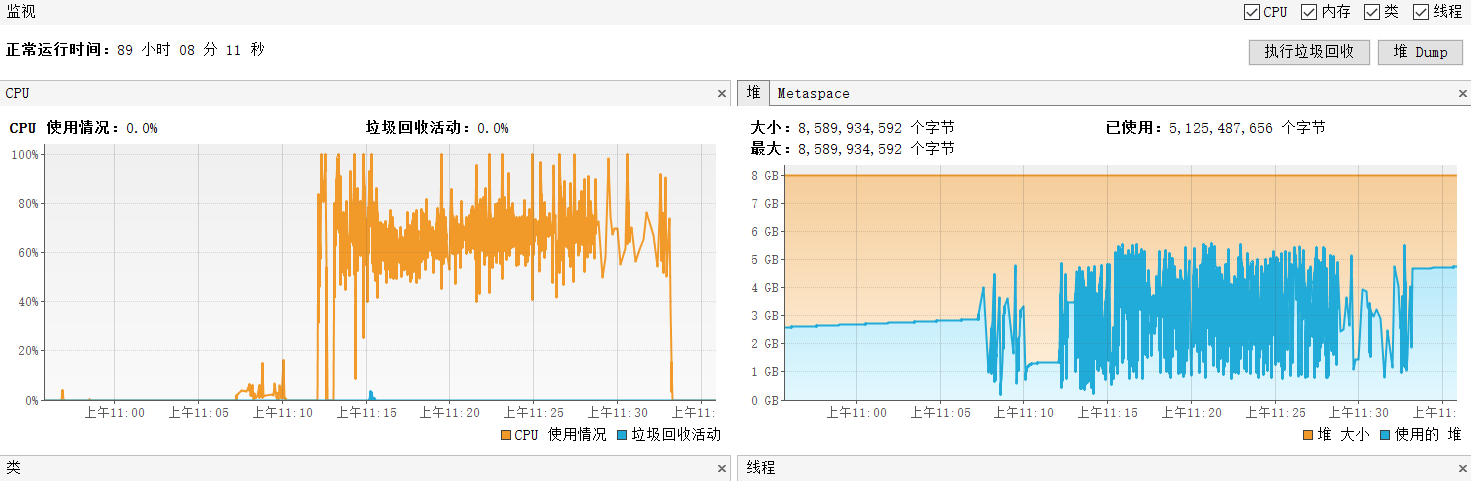
主要调优TCP内核参数，以及Linux主机突破文件句柄，单节点支持更多的连接。参考连接: <https://blog.csdn.net/fuzhongfaya/article/details/80917857>

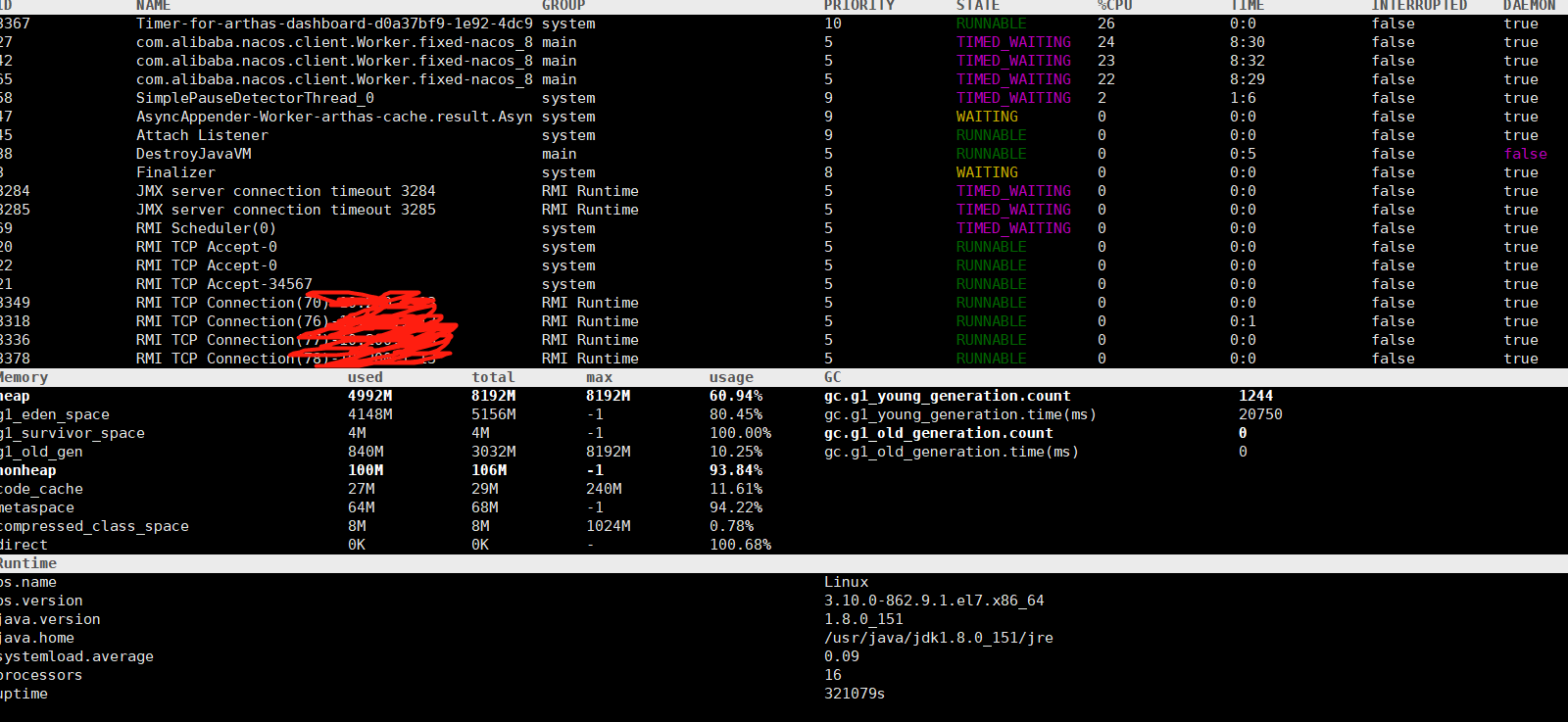
# 附录

## 808网关半裸奔参考

同样模拟25000车（以网关最佳状态模拟），只进行decode，encode。基于808协议真实传输位置信息汇报，响应平台通用应答！无任何逻辑处理。压测结果如下:







## Netty全裸奔参考

同样模拟25000车（以网关最佳状态模拟）,不进行逻辑处理，只进行decode，encode。传输4个字节，响应4个字节。最简单的传输。



