- NETTY

精尽 Netty 源码解析 —— Buffer 之 Jemalloc（一）简介

1. 概述

在看 Netty 对 Jemalloc 内存管理算法的具体代码实现之前，笔者想先通过这篇文章，**简单**阐述三个问题：

- Netty 为什么要实现内存管理?
- Netty 为什么选择 Jemalloc 算法?
- Jemalloc 的实现原理?

因为笔者对内存管理的了解程度，处于青铜级别，所以为了更好的让大家理解，本文会以引用牛 x 大佬的文章为主。

2. Netty 为什么要实现内存管理

老芳芳的理解

在 Netty 中，IO 读写必定是非常频繁的操作，而考虑到更高效的网络传输性能，Direct ByteBuffer 必然是最合适的选择。但是 Direct ByteBuffer 的申请和释放是高成本的操作，那么进行**池化**管理，多次重用是比较有效的方式。但是，不同于一般于我们常见的对象池、连接池等**池化**的案例，ByteBuffer 是有**大小**一说。又但是，申请多大的 Direct ByteBuffer 进行池化又会是一个大问题，太大会浪费内存，太小又会出现频繁的扩容和内存复制！！！所以呢，就需要有一个合适的内存管理算法，解决**高效分配内存**的同时又解决**内存碎片化**的问题。

官方的说法

> FROM 《Netty 学习笔记 —— Pooled buffer》
>
> Netty 4.x 增加了 Pooled Buffer，实现了高性能的 buffer 池，分配策略则是结合了 buddy allocation 和 slab allocation 的 **jemalloc** 变种，代码在 `io.netty.buffer.PoolArena` 中。
>
> 官方说提供了以下优势：
>
> - 频繁分配、释放 buffer 时减少了 GC 压力。
> - 在初始化新 buffer 时减少内存带宽消耗(初始化时不可避免的要给buffer数组赋初始值)。

文章目录

1. 概述

1. 概述
2. Netty 为什么要实现内存管理
3. Netty 为什么选择 Jemalloc 算法
4. Jemalloc 的实现原理
666. 彩蛋

。

> C/C++ 和 java 中有个围城，城里的想出来，城外的想进去！ **

这个围城就是自动内存管理！

Netty 4 buffer 介绍

Netty4 带来一个与众不同的特点是其 ByteBuf 的实现，相比之下，通过维护两个独立的读写指针， 要比 `io.netty.buffer.ByteBuf` 简单不少，也会更高效一些。不过，Netty 的 ByteBuf 带给我们的最大不同，就是他不再基于传统 JVM 的 GC 模式，相反，它采用了类似于 C++ 中的 malloc/free 的机制，需要开发人员来手动管理回收与释放。从手动内存管理上升到GC，是一个历史的巨大进步， 不过，在20年后，居然有曲线的回归到了手动内存管理模式，正印证了马克思哲学观： 社会总是在螺旋式前进的，没有永远的最好。

① **GC** 内存管理分析

的确，就内存管理而言，GC带给我们的价值是不言而喻的，不仅大大的降低了程序员的心智包袱， 而且，也极大的减少了内存管理带来的 Crash 困扰，为函数式编程（大量的临时对象）、脚本语言编程带来了春天。 并且，高效的GC算法也让大部分情况下程序可以有更高的执行效率。 不过，也有很多的情况，可能是手工内存管理更为合适的。譬如：

- 对于类似于业务逻辑相对简单，譬如网络路由转发型应用（很多erlang应用其实是这种类型）， 但是 QPS 非常高，比如1M级，在这种情况下，在每次处理中即便产生1K的垃圾，都会导致频繁的GC产生。 在这种模式下，erlang 的按进程回收模式，或者是 C/C++ 的手工回收机制，效率更高。
- Cache 型应用，由于对象的存在周期太长，GC 基本上就变得没有价值。

文章目录

1. 概述

1. 概述
2. Netty 为什么要实现内存管理
3. Netty 为什么选择 Jemalloc 算法
4. Jemalloc 的实现原理
666. 彩蛋

实际上比较适合于处理介于这 2 者之间的情况： 对象
理的时间要少得多的，但又是相对短暂的， 典型
处理能力在 1K QPS 量级，每个请求的对象分配在
10s 的时间内进行一 次younger GC ，每次GC的时间
可以控制在 10ms 水平上， 这类的应用，实在是太适合 GC 行的模式了，而且结合 Java 高效的分代 GC ，简直就是一个理想搭配。

② 影响

Netty 4 引入了手工内存的模式，我觉得这是一大创新，这种模式甚至于会延展， 应用到 Cache 应用中。实际上，结合 JVM 的诸多优秀特性，如果用 Java 来实现一个 Redis 型 Cache、 或者 In-memory SQL Engine，或者是一个 Mongo DB，我觉得相比 C/C++ 而言，都要更简单很多。 实际上，JVM 也已经提供了打通这种技术的机制，就是 Direct Memory 和 Unsafe 对象。 基于这个基础，我们可以像 C 语言一样直接操作内存。实际上，Netty4 的 ByteBuf 也是基于这个基础的。

3. Netty 为什么选择 Jemalloc 算法

推荐直接阅读

- bhpike65 《内存优化总结:ptmalloc、tcmalloc 和 jemalloc》

4. Jemalloc 的实现原理

推荐直接阅读

- Hypercube 《自顶向下深入分析Netty（十）–JEMalloc分配算法》
- 高兴的博客 《jemalloc和内存管里》
- 沧行 《Netty内存池实现》 这篇，有几个图，非常非常非常不错。

666. 彩蛋

推荐的博客比较多，如果你和笔者一样对内存管理的理解处于**青铜**级别，可能看完这几篇博文，很大可能还是一脸懵逼+一脸懵逼+一脸懵逼。

这是个比较正常的情况。胖友可以跟着笔者继续看看 Netty 对 Jemalloc 算法的具体实现后，再回过头继续理解下这几篇文章。

另外，后续的文章，会有大量大量大量的**位运算**，所以当胖友看到不熟悉的**位运算**，可以看看 《Java 位运算(移位、位与、或、异或、非）》 。

文章目录

次