Netty高性能编程备忘录（下）

08月 14, 2016 | Filed under 工作 (http://calvin1978.blogcn.com/articles/category/%e5%b7%a5%e4%bd%9c)

前文再续，书接上一回：Netty高性能编程备忘录（上） (http://calvin1978.blogcn.com/articles/netty-performance.html)，想不到这次这么快就写了下集，把坑填了。

3. 内存篇

3.1 堆外内存池

堆外内存是Netty被说的最多的部分，网卡内核态与应用用户态之间零复制啊，无GC啊，不受堆内存大小限制啊，不重复。
内存池的算法 (http://www.jianshu.com/p/4856bd30dd56)也是Netty骄傲的地方（注意，在4.0的刚开始版本这也是经常改的）
Norman Maurer说，只有在输出时要需要编码对象直接操作bytes[]时，才有可能用回Heap Buffer。

3.1.1 ByteBuf释放

各种异常处理，一不留神，我的踩坑之作:Netty之有效规避内存泄漏 (http://calvin1978.blogcn.com/articles/netty-leak.html)
建议写足够的单元测试，在测试里将内存泄漏检查级别开到最高，然后每个用例执行完就System.gc()一次，同时加入一个测试用的appender监控Netty的logger有没有输出memory leak信息。
如果信心已足，在生产环境里，就可以加上"-Dio.netty.leakDetectionLevel=disabled”把检测关掉，提高那么点点理论上存在的性能。

3.1.2 Recycler

Netty的另一个得意设计是对象可以在线程内无锁的被回收重用。但有些场景里，某线程只创建对象，要到另一个线程里释放，一不小心，你就会发现应用缓缓变慢，heap dump时看到好多RecyleHandler对象。所以这块的设计其实在4.0.x的各个版本里变动了无数遍，貌似4.0.40版才终于在我的测试里不再泄漏了。

但有时觉得这么辛苦的重用一个对象（而不是ByteBuffer内存本身），不如干脆禁止掉这个功能，所以4.0.0.33里，我会在启动参数里加入 -Dio.netty.recycler.maxCapacity.default=0。无语的是，也几乎从这个版本开始，才能通过设为0禁止它。

3.2 避免复制：CompositeByteBuff, slice(), duplicate()

尽量，尽量不要进行ByteBuf内容复制。

场景1: 为了失败时重试，我要保留内容稍后使用，不想Netty在发送完毕后把内容释放了，最笨的方法是用copy()复制一个新的ByteBuf。

```
Bytebuf newBuf = oldBuf.duplicate().retain();
```

而上句只是复制出独立的读写Index, 而底下的ByteBuffer是共享的，同时将ByteBuffer的计数器 + 1.

场景2: 在Proxy型的应用里，输入输出的内容不变，只替换一些头信息。

聪明的做法是，用slice().retain()语句从旧的ByteBuf中切割出Header外的Body部分，同样是共享底层ByteBuffer。然后生成一个新的Header，然后用CompositeByteBuff，将新的Header 与 旧的Body拼接起来。

3.3 避免扩容: ByteBuf的大小预估与AdaptiveRecvByteBufAllocator

ByteBuf如果一开始申请的不足，到极限时会智能的扩容，但也和Java一样，需要重新申请两倍的内存，然后把旧的内容复制过去，一听就是个很消耗的动作，因此，反正是堆外内存池，一开始还是给多一点吧。

另一个有趣的思路是Netty的自适应算法。Netty收到一个请求时，什么都不知道啊，那会申请多大的内存来接收它呢？在Bootstrap里可以配置，默认是 AdaptiveRecvByteBufAllocator，根据每一次收到的请求动态变化。

那如果一个应用有几个不同接口，请求的大小变来变去，会不会玩死它呢？好像会的。不过服务化体系里的特征都是请求小，返回大，请求包的大小变化不会太剧烈。

3.4 烦人的rangeChecking

Norman Maurer说，如果你要搜索某个Byte是否存在，请用 byteBuf.forEachByte(ByteProcessor processor), 比循环的遍历地调用byteBuf.readByte()要快得多。原因无它，ByteBuf有Java其他集合同样的rangeChecking。

每次readByte()都不是读一个字节这么简单，首先要判断refCnt()>0，然后再做范围检查防止越界。getByte(i = int)更加一层又一层的检查函数，JVM没有帮你内联或者Profiler工具阻止了你的内联的话，够呛。

3.5 readInt()，不要readBytes(bytes[],0,4)

比如Thrift，它会做一层封装，先用byteBuf.readBytes(bytes,0,4)读取byte[]，再自己转成int。

但实测证明，我将所有的读写short, int, long, boolean, byte的函数，改造为直接使用Netty的原生函数时，性能从7万QPS提升到7.4万QPS，而消耗CPU不变。

3.6 对String说不的 ASCIIString

Netty收到的bytes[]，大部分时候最终都要变回String。但String的内部是char[]啊，出入都要经过CharsetEncoder进行转换成byte[]，既浪费CPU，又浪费内存。

ByteBufUtils类提供了写入UTF-8和ASCII的优化，不需要从String创建并编码一个bytes[]再开始写入ByteBuf，而是遍历一个个char，当场编码当场写入。可惜此优化对于thrift这种需要先得到byte[]长度的编码器无效。

而Netty 4.1开始，提供了实现 CharSequence接口的ASCIIString。原理就是，String要存char[]，是因为UTF-8这样的不定长Encoder，会把char转成1～3个byte。但如果我的Header的名称与某些值，肯定是ASCII字符时（比如服务名，服务版本），那一个char只对应一个byte啊，那你直接在构造函数里把byte[]交给我内部存起来就行了啊，不需要任何转换啊，不费CPU又不废内存了啊。

4. 工具类篇

Netty 为了高效编程，或写或借，搞了一些高效的工具类，在自己的应用里同样可以借用一下。Netty自己有一篇Using as a generic library (http://netty.io/wiki/using-as-a-generic-library.html)，介绍了其中的一些。本文主要介绍与性能相关的。

4.1 FastThreadLocal

Netty威武，居然太岁头上动土，搞出个比JDK的ThreadLocal还快的ThreadLocal。详见《Netty精粹之设计更快的ThreadLocal》(http://my.oschina.net/andylucc/blog/614359)

JDK的ThreadLocal，实现原理是Thread对象里有个HashMap性质的数组，每个ThreadLocal的id是个Hashcode，算法是currentValue+0x61c88647，hashCode取模数组大小得到threadLocal存的位置，如果桶里已有其他元素，key.nextHashCode()找下一个桶，小学学过的HashMap实现之一开放地址法就不啰嗦了。

而FastThreadLocal的id则是一个自增的int，FastThreadLocalThread里放一个数组，直接按下标获取，没有hash，没有比较，没有冲突。不过需要在Netty地界里用，业务线程池就要自己定义ThreadFactory，创建FastThreadLocalThread 而不是Thread。

4.2 移植JDK8的宝贝到JDK7

JDK8重写了ConcurrentHashMap，原来的Load Factor，Current Level都没有作用了。
ThreadLocalRandom就是把原来有全局锁的Random，通过ThreadLocal化取消了锁。
LongAdder则是把AtomicLong打散成几个，平时＋＋的时候找其中一个执行，减少CMS冲突的概率，等get()的时候才把几个counter累计起来，适合increment()多，get()少的情况。

Netty把这些类都复制黏贴了一份，封装在 PlatformDependent里，根据JDK版本决定返回JDK原生的还是它复制的。

4.3 其他宝贝

4.3.1 ThreadLocal的StringBuilder

之前写过StringBuilder在高性能场景下的正确用法(http://calvin1978.blogcn.com/articles/stringbuilder.html)，才发现Netty和我做了同样的事，通过ThreadLocal的保存，重用StringBuilder对象，节约内存和分配内存的时间。当然，如果字符串只是很短就未必有必要。

4.3.2 IntHashMap

原始类型的map，比如key是int而不是Integer的Map，在某些次元里挺流行的，Trove，Koloboke, FastUtil等等，好处一是int比Integer省地方，int是4bytes，Integer是12+4 bytes，另外数据结构与解决冲突的方式也不同，详见高性能场景下，关于HashMap的补课(http://calvin1978.blogcn.com/articles/hashmap.html)

比起FastUtils.jar 穷举各种原始类型-原始类型/对象类型的组合，动不动10M大小。Netty只有IntHashMap一个类, 4.1又增加了LongHashMap等，够用了。

4.3.3 JCTools的Queue

针对Multiple Producer - Single Consumer，Single Producer - Multiple Consumer等不同场景专门设计，做到最少的锁。
不过并不提供阻塞等待的函数，所以不能拿来替换ArrayBlockingQueue。

4.3.4 RecyclableArrayList

如果你需要经常创建很长的ArrayList，不想浪费了，可以考虑用它来节约GC，不过到底哪边的代价大，一定要真正测试后决定。详见Netty精粹之轻量级内存池技术实现原理与应用(http://my.oschina.net/andylucc/blog/614589)。

5. 其他零碎篇

主要来自Norman Maurer的文章：

1. ctx.writeAndFlush() 与 channel.writeAndFlush()的区别在于，channel要经过整条Pipeline，而ctx直接找下一个outboundHandler。

2. channel.writeAndFlush(buf, channel.voidPromise())
writeAndFlush不管你用不用默认构造返回一个Promise(Future)，有点浪费内存。没有用的话，用一个公共的 voidPromise ，减少大家花费。但低版本的Netty不能用。

3. 3. 空闲连接管理，因为刚才说的ctx.writeAndFlush()可能不经过IdleHander，所以只监控读空闲就够了。而且如果每次请求都要READ/WRITE/ALL IDEL三个值算一遍，也白白消耗性能。

4. writeAndFlush()不要太多，毕竟调用了系统调用。

5. Handler能共用就标上Shareable Annotation然后共用，不要每个Channel建一个。

暂时只想到这么多。其他想起来再写吧。 最后吐槽一句，Netty即使用的再溜，你的内核参数设定，你的业务代码，其实也有很大的影响，优化时不要光盯着Netty。

转载请保留原文链接，否则视为侵权。。。。http://calvin1978.blogcn.com/articles/netty-performance2.html (http://calvin1978.blogcn.com/articles/netty-performance2.html)

有关的...

- 2016-08-14 -- Netty高性能编程备忘录(上) (http://calvin1978.blogcn.com/articles/netty-performance.html)
- 2016-10-29 -- Java性能优化指南1.8版，及唯品会的实战 (http://calvin1978.blogcn.com/articles/javatuning.html)
- 2016-10-26 -- 关键业务系统的JVM参数推荐(2016热冬版) (http://calvin1978.blogcn.com/articles/jvmoption-2.html)
- 2016-09-14 -- Btrace入门到熟练小工完全指南 (http://calvin1978.blogcn.com/articles/btrace1.html)

by calvin | tags : Netty (http://calvin1978.blogcn.com/articles/tag/netty), 调优 (http://calvin1978.blogcn.com/articles/tag/%e8%b0%83%e4%bc%98) | 2 (http://calvin1978.blogcn.com/articles/netty-performance2.html#comments)

Tomcat线程池，更符合大家想象的可扩展线程池 (http://calvin1978.blogcn.com/articles/tomcat-threadpool.html) »
« Netty高性能编程备忘录(上) (http://calvin1978.blogcn.com/articles/netty-performance.html)

2 Comments

stone
08月 16th, 2016 at 17:57 (#comment-306)

“JDK8重写了ConcurrentHashMap，原来的Load Factor，Current Level都没有了”

您文中的这句，我特意翻了jdk 1.8，还有这两个参数变量的，是我理解不对吗？

回复 (/articles/netty-performance2.html?replytocom=306#respond)

白衣
08月 17th, 2016 at 08:24 (#comment-308)

但只是參數，所起的作用甚微，成員變量已經沒了。

回复 (/articles/netty-performance2.html?replytocom=308#respond)

发表评论

您的电子邮箱不会被公开。

名称

电子邮箱

网址

评论

发表评论

分类

工作 (http://calvin1978.blogcn.com/articles/category/%e5%b7%a5%e4%bd%9c)
技术 (http://calvin1978.blogcn.com/articles/category/%e6%8a%80%e6%9c%af)
文艺 (http://calvin1978.blogcn.com/articles/category/%e6%96%87%e8%89%ba)
日常 (http://calvin1978.blogcn.com/articles/category/%e6%97%a5%e5%b8%b8)
议论 (http://calvin1978.blogcn.com/articles/category/%e8%ae%ae%e8%ae%ba)

归档

2017年10月 (http://calvin1978.blogcn.com/articles/2017/10)
2016年11月 (http://calvin1978.blogcn.com/articles/2016/11)
2016年10月 (http://calvin1978.blogcn.com/articles/2016/10)
2016年09月 (http://calvin1978.blogcn.com/articles/2016/09)
2016年08月 (http://calvin1978.blogcn.com/articles/2016/08)
2016年07月 (http://calvin1978.blogcn.com/articles/2016/07)
2016年05月 (http://calvin1978.blogcn.com/articles/2016/05)
2016年04月 (http://calvin1978.blogcn.com/articles/2016/04)
2016年01月 (http://calvin1978.blogcn.com/articles/2016/01)
2015年12月 (http://calvin1978.blogcn.com/articles/2015/12)
2015年11月 (http://calvin1978.blogcn.com/articles/2015/11)
2015年09月 (http://calvin1978.blogcn.com/articles/2015/09)

链接

RSS (/feed)
我的微博 (http://www.weibo.com/calvin1978)
春天的旁边 (http://springside.io/)

标签云

AboutMe (http://calvin1978.blogcn.com/articles/tag/aboutme)
bigdata (http://calvin1978.blogcn.com/articles/tag/bigdata)
Dolphin (http://calvin1978.blogcn.com/articles/tag/dolphin)
Football (http://calvin1978.blogcn.com/articles/tag/football)
java (http://calvin1978.blogcn.com/articles/tag/java)
Netty (http://calvin1978.blogcn.com/articles/tag/netty)
Redis (http://calvin1978.blogcn.com/articles/tag/redis)
SpringSide (http://calvin1978.blogcn.com/articles/tag/springside)
SpringSide AboutMe (http://calvin1978.blogcn.com/articles/tag/springside-aboutme)
云门舞集 (http://calvin1978.blogcn.com/articles/tag/%e4%ba%91%e9%97%a8%e8%88%9e%e9%9b%86)
人山人海 (http://calvin1978.blogcn.com/articles/tag/%e4%ba%ba%e5%b1%b1%e4%ba%ba%e6%b5%b7)
伍迪艾伦 (http://calvin1978.blogcn.com/articles/tag/%e4%bc%8d%e8%bf%aa%e8%89%be%e4%bc%a6)
卡夫卡 (http://calvin1978.blogcn.com/articles/tag/%e5%8d%a1%e5%a4%ab%e5%8d%a1)
古诗 (http://calvin1978.blogcn.com/articles/tag/%e5%8f%a4%e8%af%97)
在节假日 (http://calvin1978.blogcn.com/articles/tag/%e5%9c%a8%e8%8a%82%e5%81%87%
技术会 (http://calvin1978.blogcn.com/articles/tag/%e6%8a%80%e6%9c%af%e4%bc%9a)
方所 (http://calvin1978.blogcn.com/articles/tag/%e6%96%b9%e6%89%80)
服务化 (http://calvin1978.blogcn.com/articles/tag/%e6%9c%8d%e5%8a%a1%e5%8c%96)
现代舞 (http://calvin1978.blogcn.com/articles/tag/%e7%8e%b0%e4%bb%a3%e8%88%9e)
现代诗 (http://calvin1978.blogcn.com/articles/tag/%e7%8e%b0%e4%bb%a3
知识分子 (http://calvin1978.blogcn.com/articles/tag/%e7%9f%a5%e8%af%86%e5%88%86%e5%ad%90)
码农

(http://calvin1978.blogcn.com/articles/tag/%e7%a0%81%e5%86%9c)
窦唯
(http://calvin1978.blogcn.com/articles/tag/%e7%aa%a6%e5%94%af)
调优
(http://calvin1978.blogcn.com/articles/tag/%e8%b0%83%e4%bc%98)
音乐现场
(http://calvin1978.blogcn.com/articles/tag/%e9%9f%b3%e4%b9%90%e7%8e%b0%e5%9c%l
黄耀明
(http://calvin1978.blogcn.com/articles/tag/%e9%bb%84%e8%80%80%e6%98%8e)