# 剖析Disruptor:为什么会这么快？（一）Ringbuffer的特别之处

原文地址：http://ifeve.com/ringbuffer/

**作者**：Trisha　**译者**：寒桐　**校对**：方腾飞

最近，我们开源了LMAX Disruptor，它是我们的交易系统吞吐量快（LMAX是一个新型的交易平台，号称能够单线程每秒处理数百万的订单）的关键原因。为什么我们要将其开源？我们意识到对高性能编程领域的一些传统观点，有点不对劲。我们找到了一种更好、更快地在线程间共享数据的方法，如果不公开于业界共享的话，那未免太自私了。同时开源也让我们觉得看起来更酷。

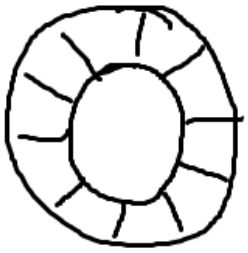
从这个站点，你可以下载到一篇解释什么是Disruptor及它为什么如此高性能的文档。这篇文档的编写过程，我并没有参与太多，只是简单地插入了一些标点符号和重组了一些我不懂的句子，但是非常高兴的是，我仍然从中提升了自己的写作水平。

我发现要把所有的事情一下子全部解释清楚还是有点困难的，所有我准备一部分一部分地解释它们，以适合我的NADD听众。

首先介绍ringbuffer。我对Disruptor的最初印象就是ringbuffer。但是后来我意识到尽管ringbuffer是整个模式（Disruptor）的核心，但是Disruptor对ringbuffer的访问控制策略才是真正的关键点所在。

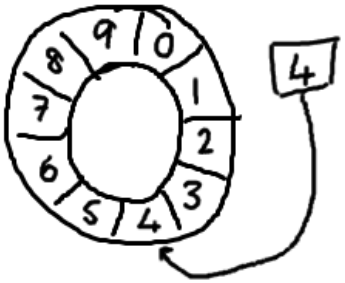
## ringbuffer到底是什么？

嗯，正如名字所说的一样，它是一个环（首尾相接的环），你可以把它用做在不同上下文（线程）间传递数据的buffer。

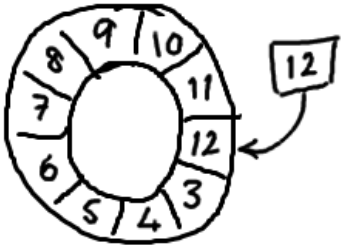


（好吧，这是我通过画图板手画的，我试着画草图，希望我的强迫症不会让我画完美的圆和直线）

基本来说，ringbuffer拥有一个序号，这个序号指向数组中下一个可用的元素。*（校对注：如下图右边的图片表示序号，这个序号指向数组的索引4的位置。）*




随着你不停地填充这个buffer（可能也会有相应的读取），这个序号会一直增长，直到绕过这个环。




要找到数组中当前序号指向的元素，可以通过mod操作：

> sequence mod array length = array index

以上面的ringbuffer为例（java的mod语法）：12 % 10 = 2。很简单吧。

事实上，上图中的ringbuffer只有10个槽完全是个意外。如果槽的个数是2的N次方更有利于基于二进制的计算机进行计算。

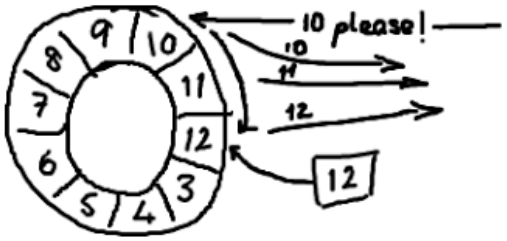
*（校对注：2的N次方换成二进制就是1000，100，10，1这样的数字， sequence &（array length - 1）= array index，比如一共有8槽，3&（8 - 1）=3，HashMap就是用这个方式来定位数组元素的，这种方式比取模的速度更快。）*

# 那又怎么样？

如果你看了维基百科里面的关于环形buffer的词条，你就会发现，我们的实现方式，与其最大的区别在于：没有尾指针。我们只维护了一个指向下一个可用位置的序号。这种实现是经过深思熟虑的—我们选择用环形buffer的最初原因就是想要提供可靠的消息传递。我们需要将已经被服务发送过的消息保存起来，这样当另外一个服务通过nak (校对注：拒绝应答信号)告诉我们没

有成功收到消息时，我们能够重新发送给他们。

听起来，环形buffer非常适合这个场景。它维护了一个指向尾部的序号，当收到nak(校对注：拒绝应答信号)请求，可以重发从那一点到当前序号之间的所有消息：




我们实现的ring buffer和大家常用的队列之间的区别是，我们不删除buffer中的数据，也就是说这些数据一直存放在buffer中，直到新的数据覆盖他们。这就是和维基百科版本相比，我们不需要尾指针的原因。ringbuffer本身并不控制是否需要重叠（决定是否重叠是生产者-消费者行为模式的一部分–如果你等不急我写blog来说明它们，那么可以自行检出Disruptor项目）。

# 它为什么如此优秀？

之所以ringbuffer采用这种数据结构，是因为它在可靠消息传递方面有很好的性能。这就够了，不过它还有一些其他的优点。

首先，因为它是数组，所以要比链表快，而且有一个容易预测的访问模式。（*译者注：数组内元素的内存地址的连续性存储的*）。这是对CPU缓存友好的—也就是说，在硬件级别，数组中的元素是会被预加载的，因此在ringbuffer当中，cpu无需时不时去主存加载数组中的下一个元素。*（校对注：因为只要一个元素被加载到缓存行，其他相邻的几个元素也会被加载进同一个缓存行）*

其次，你可以为数组预先分配内存，使得数组对象一直存在（除非程序终止）。这就意味着不需要花大量的时间用于垃圾回收。此外，不像链表那样，需要为每一个添加到其上面的对象创造节点对象—对应的，当删除节点时，需要执行相应的内存清理操作。

# 缺少的部分

我并没有在本文中介绍如何避免ringbuffer产生重叠，以及如何对ringbuffer进行读写操作。你可能注意到了我将ringbuffer和链表那样的数据结构进行比较，因为我并认为链表是实际问题的标准答案。

当你将Disruptor和基于 队列之类的实现进行比较时，事情将变得很有趣。队列通常注重维护队列的头尾元素，添加和删除元素等。所有的这些我都没有在ringbuffer里提到，这是因为ringbuffer不负责这些事情，我们把这些操作都移到了数据结构（ringbuffer）的外部

到这个站点阅读文章或者检出代码可以了解更多细节。或者观看Mike 和Martin在去年San Francisco QCon大会上的视频，或者再等我一些时间来思考剩下的东西，然后在接下来的blog中逐一介绍。

About 。Latest Posts

**Olylakers**

欧立勇。花名寒桐，阿里巴巴资深开发工程师。

发表评论 RSS订阅评论 Trackback 关闭 评论 (14)

夕水溪下
2013/02/01 3:34下午

登录以回复 | 引用

RingBuffer的特别之处是他没有尾指针，整个数据结构中只有一个变量，用这个变量来决定可用的存储位置。当然，这个变量就是共享变量了，我们需要同步来保证原子性，而在disruptor中使用了volatile来保证线程安全，效率更高。如此简单的数据结构，我们无法得知队列中哪些数据已经被消费，哪些数据没有被消费，这些信息都在RingBuffer之外保存，比如消费者会保存他目前的"消费指数"。

Atry
2013/02/28 10:02下午

登录以回复 | 引用

这系统不是号称单线程每秒600万次事务吗？那这样算的话，每次事务就是一千六百多纳秒。可CPU每次即使L2缓存未命中，也不过浪费几十纳秒从主内存加载而已。所以用什么数组代替链表也就为每次事务省了几十纳秒，几乎可以忽略不计。

Atry
2013/02/28 10:03下午

登录以回复 | 引用

再说了，用Java随便写几行逻辑处理代码就会导致一大堆内存分配。这些内存分配都是在堆上呢，早就把CPU缓存给破坏了。真想做内存局部性的优化，趁早别用Java.

kiwi
2013/03/26 10:18上午

登录以回复 | 引用

只写几行代码的人根本就不关心内存。java的垃圾回收优秀之处在于开发基本上不需要过问gc！java效率确实低于c++之类，但是鱼与熊掌不可兼得，没必要非得诋毁哪一种！

方 腾飞
2013/03/18 1:01上午

登录以回复 | 引用

Ringbuffer的特别之处。

1：减少竞争点，比如不删除数据，所以不需要尾指针。

2：重复利用数组，减少GC。

3: 使用数组存储数据，可以利用缓存每次都加载一个cacheline的特性。

nicky
2013/03/24 10:59上午

登录以回复 | 引用

想弄明白以下几点

1 ringbuffer的序号是如何维护的？ 初始化的时候序号是指向0的么，当往ringbuffer增加元素，序号会变化么，读取元素的时候，会增加序号么？

2 如果一次输入的元素个数大于ringbuffer的长度，那不是有元素在没有被读取的时候就被覆盖了么

3 为什么当length 为2的N次方 sequence &(length – 1 ) = sequence%length ?

方 腾飞
2013/03/26 1:25上午

登录以回复 | 引用

1:后面几篇文章有详细介绍ringbuffer。

2:不会覆盖，会等待。

3:不相等，只是二进制与比取模更快。但是二进制与需要数据为2的N次方才行。

manu_1984
2014/06/28 9:56上午

登录以回复 | 引用

读取元素的时候，会增加序号么？

xumingmingv
2013/03/27 10:40下午

登录以回复 | 引用

“因为我并认为链表是实际问题的标准答案。”应该是：“因为我并不认为有人会相信链表是世界上问题的标准答案。”

liuinsect
2014/03/11 1:26下午

登录以回复 | 引用

为什么没有人说翻译错误的问题？

比如：“因为我并认为链表是实际问题的标准答案”这句话读起来流畅么？

jack
2014/03/20 5:05下午

登录以回复 | 引用

rignbuffer的数据如果被填充满了，后续的数据进不来，怎么处理

> 天小陈
> 2014/09/12 4:44下午
>
> 登录以回复 | 引用
>
> 我也认为这是一个问题。

E网情深
2014/12/14 12:09下午

登录以回复 | 引用

> jack :
>
> rignbuffer的数据如果被填充满了，后续的数据进不来，怎么处理

被填冲慢了，只要有消费，就可以把之前数据更新掉;

所以RingBuffer在插入元素时，是2阶段提交，首先申请sequence,如果发现所有数据都没被被消费，会等待

LockSupportparkNan(),直到有可用的seq,获取得可以用的seq后，再publish

E网情深
2014/12/14 12:12下午

登录以回复 | 引用

> 方腾飞 :

1:后面几篇文章有详细介绍ringbuffer。

2:不会覆盖，会等待。

3:不相等，只是二进制与比取模更快。但是二进制与需要数据为2的N次方才行。

为什么当length 为2的N次方 sequence &(length – 1 ) = sequence%length ？，是2的n次方时，应该相等吧，