

欢迎光临汤雪华的博客

一个人一辈子能坚持做好一件事情就够了！坚持是一种刻意的练习，不断寻找缺点突破缺点的过程，而不是重复做某件事情。

博客园

首页

新随笔

联系

订阅

管理

随笔 - 130 文章 - 0 评论 - 1907

公告

EQueue - 一个C#写的开源分布式消息队列的总体介绍

大家好，我叫汤雪华，我主要专注的领域是DDD/CQRS。目前致力于我的两个个人开源项目：**ENode**、**EQueue**。欢迎大家扫描下面的二维码关注我的微信公众号：**ENode**。我会定期分享我的心得体会，也欢迎加入QQ群（185916873）进行交流。



昵称: netfocus
园龄: 10年7个月
荣誉: 推荐博客
粉丝: 2111
关注: 21
[+加关注](#)

搜索

找找看

谷歌搜索

随笔分类(139)

- CQRS & Event Sourcing(11)
- DDD 案例分析(14)
- DDD 理论积累(22)
- ENode(22)
- EQueue(10)
- 好文摘录(19)
- 架构(17)
- 面向对象(4)
- 杂项(11)
- 哲学(4)
- 蜘蛛侠论坛(5)

随笔档案(130)

2016年9月 (3)

前言

本文想介绍一下前段时间在写enode时，顺便实现的一个分布式消息队列equeue。这个消息队列的思想不是我想出来的，而是通过学习阿里的rocketmq后，自己用c#实现了一个轻量级的简单版本。一方面可以通过写这个队列让自己更深入的掌握消息队列的一些常见问题；另一方面也可以用来和enode集成，为enode中的command和domain event的消息传递提供支持。目前在.net平台，比较好用的消息队列，最常见的是微软的MSMQ了吧，还有像rabbitmq也有.net的client端。这些消息队列都很强大和成熟。但当我学习了kafka以及阿里的rocketmq（早期版本叫metaq，自metaq 3.0后改名为rocketmq）后，觉得rocketmq的设计思想深深吸引了我，因为我不仅能理解其思想，还有其完整的源代码可以学习。但是rocketmq是java写的，且目前还没有.net的client端，所以不能直接使用（有兴趣的朋友可以为其写一个.net的client端），所以在学习了rocketmq的设计文档以及大部分代码后，决定自己用c#写一个出来。

项目开源地址：<https://github.com/tangxuehua/equeue>，项目中包含了队列的全部源代码以及如何使用的示例。也可以在enode项目中看到如何使用。

EQueue消息队列中的专业术语

Topic

一个topic就是一个主题。一个系统中，我们可以对消息划分为一些topic，这样我们就能通过topic，将消息发送到不同的queue。

Queue

一个topic下，我们可以设置多个queue，每个queue就是我们平时所说的消息队列；因为queue是完全从属于某个特定的topic的，所以当我们要发送消息时，总是要指定该消息所属的topic是什么。然后equeue就能知道该topic下有几个queue了。但是到底发送到哪个queue呢？比如一个topic下有4个queue，那对于这个topic下的消息，发送时，到底该发送到哪个queue呢？那必定有个消息被路由的过程。目前equeue的做法是在发送一个消息时，需要用户指定这个消息对应的topic以及一个用来路由的一个object类型的参数。equeue会根据topic得到所有的queue，然后根据该object参数通过hash code然后取模queue的个数最后得到要发送的queue的编号，从而知道该发送到哪个queue。这个路由消息的过程是在发送消息的这一方做的，也就是下面要说的producer。之所以不在消息服务器上做是因为这样可以让用户自己决定该如何路由消息，具有更大的灵活性。

Producer

就是消息队列的生产者。我们知道，消息队列的本质就是实现了publish-subscribe的模式，即生产者-消费者模式。生产者生产消息，消费者消费消息。所以这里的Producer就是用来生产和发送消息的。

Consumer

就是消息队列的消费者，一个消息可以有多个消费者。

Consumer Group

消费者分组，这可能对大家来说是一个新概念。之所以要搞出一个消费者分组，是为了实现下面要说的集群消费。一个消费者分组中包含了一些消费者，如果这些消费者是要集群消费，那这些消费者会平均消费该分组中的消息。

Broker

equeue中的broker负责消息的中转，即接收producer发送过来的消息，然后持久化消息到磁盘，然后接收consumer发送过来的拉取消息的请求，然后根据请求拉取相应的消息给consumer。所以，broker可以理解为消息队列服务器，提供消息的接收、存储、拉取服务。可见，broker对于equeue来说是核

心，它绝对不能挂，一旦挂了，那producer，consumer就无法实现publish-subscribe了。

集群消费

集群消费是指，一个consumer group下的consumer，平均消费topic下的queue。具体如何平均可以看一下下面的架构图，这里先用文字简单描述一下。假如一个topic下有4个queue，然后当前有一个consumer group，该分组下有4个consumer，那每个consumer就被分配到该topic下的一个queue，这样就达到了平均消费topic下的queue的目的。如果consumer group下只有两个consumer，那每个consumer就消费2个queue。如果有3个consumer，则第一个消费2个queue，后面两个每个消费一个queue，从而达到尽量平均消费。所以，可以看出，我们应该尽量让consumer group下的consumer的数目和topic的queue的数目一致或成倍数关系。这样每个consumer消费的queue的数量总是一样的，这样每个consumer服务器的压力才会差不多。当前前提是这个topic下的每个queue里的消息的数量总是差不多多的。这点我们可以对消息根据某个用户自己定义的key来进行hash路由来保证。

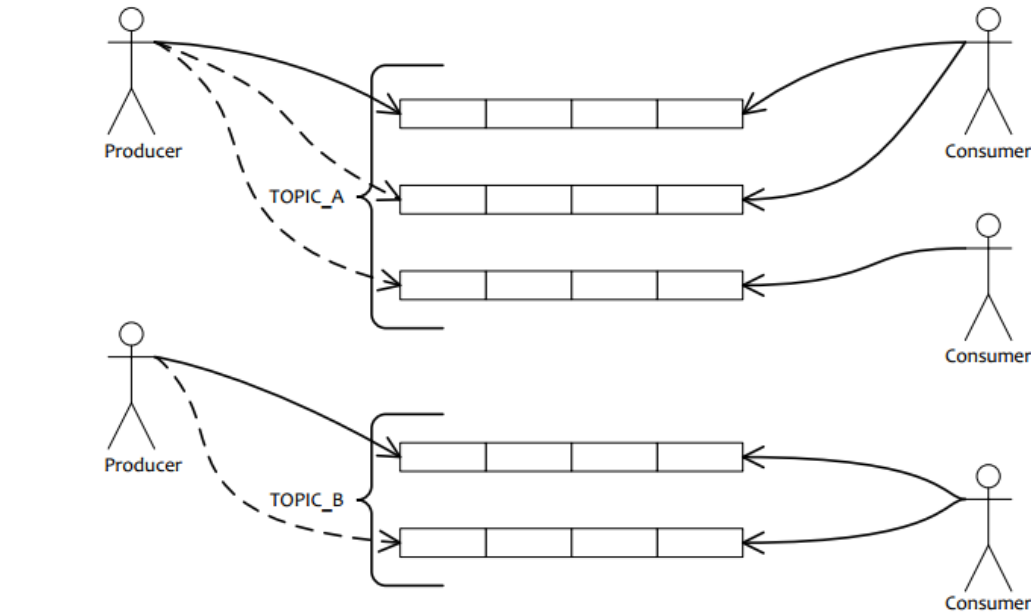
广播消费

广播消费是指一个consumer只要订阅了某个topic的消息，那它就会收到该topic下的所有queue里的消息，而不管这个consumer的group是什么。所以对于广播消费来说，consumer group没什么实际意义。consumer可以在实例化时，我们可以指定是集群消费还是广播消费。

消费进度(offset)

消费进度是指，当一个consumer group里的consumer在消费某个queue里的消息时，equeue是通过记录消费位置(offset)来知道当前消费到哪里了。以便该consumer重启后继续从该位置开始消费。比如一个topic有4个queue，一个consumer group有4个consumer，则每个consumer分配到一个queue，然后每个consumer分别消费自己的queue里的消息。equeue会分别记录每个consumer对其queue的消费进度，从而保证每个consumer重启后知道下次从哪里开始继续消费。实际上，也许下次重启后不是由该consumer消费该queue了，而是由group里的其他consumer消费了，这样也没关系，因为我们已经记录了这个queue的消费位置了。所以可以看出，消费位置和consumer其实无关，消费位置完全是queue的一个属性，用来记录当前被消费到哪里了。另外一点很重要的是，一个topic可以被多个consumer group里的consumer订阅。不同consumer group里的consumer即便是消费同一个topic下的同一个queue，那消费进度也是分开存储的。也就是说，不同的consumer group内的consumer的消费完全隔离，彼此不受影响。还有一点就是，对于集群消费和广播消费，消费进度持久化的地方是不同的，集群消费的消费进度是放在broker，也就是消息队列服务器上的，而广播消费的消费进度是存储在consumer本地磁盘上的。之所以这样设计是因为，对于集群消费，由于一个queue的消费者可能会更换，因为consumer group下的consumer数量可能会增加或减少，然后就会重新计算每个consumer该消费的queue是哪些，这个能理解的把？所以，当出现一个queue的consumer变动的时候，新的consumer如何知道该从哪里开始消费这个queue呢？如果这个queue的消费进度是存储在前一个consumer服务器上的，那就很难拿到这个消费进度了，因为有可能那个服务器已经挂了，或者下架了，都有可能。而因为broker对于所有的consumer总是在服务的，所以，在集群消费的情况下，被订阅的topic的queue的消费位置是存储在broker上的，存储的时候按照不同的consumer group做隔离，以确保不同的consumer group下的consumer的消费进度互补影响。然后，对于广播消费，由于不会出现一个queue的consumer会变动的情况，所以我们没必要让broker来保存消费位置，所以是保存在consumer自己的服务器上。

EQueue是什么？



- 2016年8月 (1)
- 2016年7月 (2)
- 2016年6月 (1)
- 2016年5月 (1)
- 2016年4月 (1)
- 2016年3月 (1)
- 2016年2月 (4)
- 2016年1月 (1)
- 2015年11月 (2)
- 2015年9月 (1)
- 2015年8月 (2)
- 2015年7月 (1)
- 2015年6月 (12)
- 2015年5月 (8)
- 2015年4月 (2)
- 2015年2月 (1)
- 2015年1月 (1)
- 2014年12月 (2)
- 2014年11月 (1)
- 2014年10月 (1)
- 2014年9月 (1)
- 2014年7月 (5)
- 2014年6月 (1)
- 2014年4月 (2)
- 2014年3月 (5)
- 2014年1月 (1)
- 2013年10月 (2)
- 2013年9月 (2)
- 2013年8月 (1)
- 2013年7月 (4)
- 2013年6月 (6)
- 2013年5月 (1)
- 2013年4月 (2)
- 2013年3月 (4)
- 2013年2月 (4)
- 2012年10月 (1)
- 2012年9月 (5)
- 2012年2月 (11)
- 2012年1月 (1)
- 2011年12月 (2)
- 2011年11月 (2)
- 2011年10月 (1)
- 2011年9月 (2)
- 2011年7月 (1)
- 2011年6月 (1)
- 2011年4月 (2)
- 2011年3月 (2)
- 2011年1月 (1)
- 2010年12月 (2)
- 2010年8月 (1)
- 2010年4月 (1)
- 2010年3月 (2)
- 2010年1月 (2)

积分与排名

积分 - 330822
排名 - 383

最新评论

- 1. Re:EQueue 2.0 - 第一个真实案例剖析-一个简易论坛（Forum）
楼主你好。我在github 上下载了源代码，你说要。修改Forum.BrokerService、Forum.CommandService、Forum.EventService、Forum.Web、Fo.....
--流星雨9015
- 2. Re:EQueue 2.0 性能测试报告

通过上图，我们能直观的理解equeue。这个图是从rocketmq的设计文档中拿来的，呵呵。由于equeue的设计思想完全和rocketmq一致，所以我就拿来用了。每个producer可以向某个topic发消息，发送的时候根据某种路由策略（producer可自定义）将消息发送到某个特定的queue。然后consumer可以消费特定topic下的queue里的消息。上图中，TOPIC_A有两个消费者，这两个消费者是在一个group里，所以应该平均消费TOPIC_A下的queue但由于有三个queue，所以第一个consumer分到了2个queue，第二个consumer分到了1个。对于TOPIC_B，由于只有一个消费者，那TOPIC_B下的所有queue都由它消费。所有的topic信息、queue信息、还有消息本身，都存储在broker服务器上。这点上图中没有体现出来。上图主要关注producer,consumer,topic,queue这四个东西之间的关系，并不关注物理服务器的部署架构。

关键问题的思考

1.Producer,Broker,Consumer三者之间如何通信


由于是用c#实现，且因为一般是在局域网内部署，为了实现高性能通信，我们可以利用异步socket来通信。.net本身提供了很好的异步socket通信的支持；我们也可以用zeromq来实现高性能的socket通信。本来想直接使用zeromq来实现通信模块就好了，但后来自己学习了一下.net自带的socket通信相关知识，发现也不难，所以就自己实现了一个，呵呵。自己实现的好处是我可以自己定义消息的协议，目前这部分实现代码在ecommon基础类库中，是一个独立的可服用的与业务场景无关的基础类库。有兴趣的可以去下载来看看代码。经过了自已的一些性能测试，发现通信模块的性能还是不错的。一台broker，四台producer同时向这个broker发送消息，每秒能发送的消息4W没有问题，更多的producer还没测试。

2.消息如何持久化

消息持久化方面主要考虑的是性能问题，还有就是消息如何快速的读取。

1. 首先，一台broker上的消息不需要一直保存在该broker服务器上，因为这些消息总会被消费掉。根据阿里rocketmq的设计，默认会1天删除一次已经被消费过的消息。所以，我们可以理解，broker上的消息应该不会无限制增长，因为会被定期删除。所以不必考虑一台broker上消息放不下的问题。
2. 如何快速的持久化消息？一般来说，我觉得有两种方式：1）顺序写磁盘文件；2）用现成的key,value的nosql产品来存储；rocketmq目前用的是自己写文件的方式，这种方式的难点是写文件比较复杂，因为所有消息都是顺序append到文件末尾，虽然性能非常高，但复杂度也很高；比如所有消息不能全写在一个文件里，一个文件到达一定大小后需要拆分，一旦拆分就会产生很多问题，呵呵。拆分后如何读取也是比较复杂的问题。还有由于是顺序写入文件的，那我们还需要把每一个消息在文件中的起始位置和长度需要记录下来，这样consumer在消费消息时，才能根据offset从文件中拿到该消息。总之需要考虑的问题很多。如果是用nosql来持久化消息，那可以省去我们写文件时遇到的各种问题，我们只需要关心如何把消息的key和该消息在queue中的offset对应起来即可。另外一点疑问是，queue里的信息要持久化吗？先要想清楚queue里放的是什么东西。当broker接收到一个消息后，首先肯定是要先持久化，完成后需要把消息放入queue里。但由于内存很有限，我们不可能把这个消息直接放入queue里，我们其实要放的只需要时该消息在nosql里的key即可，或者如果是用文件来持久化，那放的是该消息在文件中的偏移量offset，即存储在文件的那个位置（比如是哪个行号）。所以，实际上，queue只是一个消息的索引。那有必要持久化queue吗？可以持久化，这样毕竟在broker重启的时候，恢复queue的时间可以缩短。那需要和持久化消息同步持久化吗？显然不需要，我们可以异步定时持久化每个queue，然后恢复queue的时候，可以先从持久化的部分恢复，然后再把剩下的部分通过持久化的消息来补充以达到queue因为异步持久化而慢的部分可以追平。所以，经过上面的分析，消息本身都是放在nosql中，queue全部在内存中。

那消息如何持久化呢？我觉得最好的办法是让每个消息有一个全局的序号，一旦消息被写入nosql后，该消息的全局序号就确定了，然后我们在更新对应的queue的信息时，把该消息的全局序号传给queue，这样queue就能把queue自己对该消息的本地序号和该消息的全局序号建立映射关系。相关代码如下：



```
public MessageStoreResult StoreMessage(Message message, int queueId)
{
    var queues = GetQueues(message.Topic);
    var queueCount = queues.Count;
    if (queueId >= queueCount || queueId < 0)
    {
        throw new InvalidQueueIdException(message.Topic, queueCount, queueId);
    }
    var queue = queues[queueId];
    var queueOffset = queue.IncrementCurrentOffset();
    var storeResult = _messageStore.StoreMessage(message, queue.QueueId, queueOffset);
    queue.SetMessageOffset(queueOffset, storeResult.MessageOffset);
    return storeResult;
}
```

这么差的机器，测试这么多的客户端，测试结论没意义。你实际只有300个，测试用了5000个，差太多了吧，多了一个数量级多。

--netfocus

3. Re:EQueue 2.0 性能测试报告
5000个客户端是模拟TCP通信访问的一个socket服务，这个服务会只有一个生产者写消息队列。不过电脑确实很差，实际的环境是300个客户端连接、访问socket，然后写消息队列。

--鸟瞻人生

4. Re:EQueue 2.0 性能测试报告
而且这么差的机器，还要搞5000个客户端？显然不行的呀。另外为何你要模拟5000个客户端，实际生产环境，一台机器一个客户端。你一个应用的集群总共才几台机器而已呀。

--netfocus

5. Re:EQueue 2.0 性能测试报告
消息队列服务器的配置至少8核16G内存的。

--netfocus

6. Re:EQueue 2.0 性能测试报告
这么差的机器，不用测试了，2G的机器能做什么呀。

--netfocus

7. Re:EQueue 2.0 性能测试报告
我的邮箱383770193@qq.com 求大神解答

--鸟瞻人生

8. Re:EQueue 2.0 性能测试报告
这个超时的情况如何解决，另外EQueue.AdminWeb也经常出现BrokerServer服务突然消失的情况

--鸟瞻人生

9. Re:EQueue 2.0 性能测试报告
测试场景1台PC机器：QuickStart.NameServer、QuickStart.BrokerServer1、QuickStart.BrokerServer2、三个服务存在在这太机器，不过这台机.....

--鸟瞻人生

10. Re:EQueue 2.0 性能测试报告
2016-10-20 16:35:31,641 ERROR ECommon.Scheduling.ScheduleService - Task has exception, name: Refresh.....

--鸟瞻人生

11. Re:浅谈我对DDD领域驱动设计的理解
就喜欢这种通俗易懂的文章

--蒋先生

12. Re:ENode简介与各种教学视频资源汇总（要进群这篇文章必看）
能否在每个视频的后面，加上录制的日期？

--lcs-帅

13. Re:DDD领域驱动设计基本理论知识总结
为什么不把工作单元放在领域内部？应用服务不应该承担这些吧。还有我在想，从仓储中创建一个领域相当的别组。

--有容乃大

14. Re:EQueue 2.3.2版本发布（支持高可用）
@雷兽1）数据不丢是叫数据高可靠，高可用是指broker服务的高可用。2）broker group的主挂了，consumer完全可以从broker从消费，不会造成次数问题。...

1. 浅谈12306核心模型设计思路和架构设计(87)
2. EQueue - 一个C#写的开源分布式消息队列的总体介绍(85)
3. 谈一下关于CQRS架构如何实现高性能(79)
4. 关于领域驱动设计（DDD）中聚合设计的一些思考(78)
5. DDD领域驱动设计基本理论知识总结(78)
6. 分享我对代码命名的一点思考和理解(74)
7. ENode 2.0 - 第一个真实案例剖析-一个简易论坛（Forum）(60)
8. CQRS\ES架构介绍(54)
9. C#分布式消息队列 EQueue 2.0 发布啦(53)
10. 领域驱动设计（DDD）部分核心概念的个人理解(52)



没什么比代码更能说明问题了，呵呵。上的代码的思路是，接收一个消息对象和一个queueId，queueId表示当前消息要放到第几个queue里。然后内部逻辑是，先获取该消息的topic的所有queue，由于queue和topic都在内存，所以这里没性能问题。然后检查一下当前传递进来的queueId是否合法。如果合法，那就定位到该queue，然后通过IncrementCurrentOffset方法，将queue的内部序号加1并返回，然后持久化消息，持久化的时候把queueId以及queueOffset一起持久化，完成后返回一个消息的全局序列号。由于messageStore内部会把消息内容、queueId、queueOffset，以及消息的全局顺序号一起作为一个整体保存到nosql中，key就是消息的全局序列号，value就是前面说的整体（被序列化为二进制）。然后，在调用queue的SetMessageOffset方法，把queueOffset和message的全局offset建立映射关系即可。最后返回一个结果。messageStore.StoreMessage的内存实现大致如下：



```
public MessageStoreResult StoreMessage(Message message, int queueId, long queueOffset)
{
    var offset = GetNextOffset();
    _queueCurrentOffsetDict[offset] = new QueueMessage(message.Topic, message.Body,
offset, queueId, queueOffset, DateTime.Now);
    return new MessageStoreResult(offset, queueId, queueOffset);
}
```



GetNextOffset就是获取下一个全局的消息序列号，QueueMessage就是上面所说的“整体”，因为是内存实现，所以就用了ConcurrentDictionary来保存一下queueMessage对象。如果是用nosql来实现messageStore，则这里需要写入nosql，key就是消息的全局序列号，value就是queueMessage的二进制序列化数据。通过上面的分析我们可以知道我们会将消息的全局序列号+queueId+queueOffset一起整体作为一条记录持久化起来。这样做有两个非常好的特性：1）实现了消息持久化和消息在queue中的位置的持久化的原子事务；2）我们总是可以根据这些持久化的queueMessage还原出所有的queue的信息，因为queueMessage里包含了消息和消息在queue的中的位置信息；

基于这样的消息存储，当某个consumer要消费某个位置的消息时，我们可以通过先通过queueId找到queue，然后通过消息在queueOffset(由consumer传递过来的)获取消息的全局offset，然后根据该全局的offset作为key从nosql拿到消息。实际上现在的equeue是批量拉取消息的，也就是一次socket请求不是拉一个消息，而是拉一批，默认是32个消息。这样consumer可以用更少的网络请求拿到更多的消息，可以加快消息消费的速度。

3.Producer发送消息时的消息路由的细节

producer在发送消息时，如何知道当前topic下有多少个queue呢？每次发送消息时都要去broker上查一下吗？显然不行，这样发送消息的性能就上不去了。那怎么办呢？就是异步，呵呵。producer可以定时向broker发送请求，获取topic下的queue数量，然后保存起来。这样每次producer在发送消息时，就只要从本地缓存里拿即可。因为broker上topic的queue的数量一般不会变化，所以这样的缓存很有意义。那还有一个问题，当前producer第一次对某个topic发送消息时，queue哪里来呢？因为定时线程不知道要向broker拿哪个topic下的queue数量，因为此时producer端还没有一个topic呢，因为一个消息都还没发送过。那就是需要判断一下，如果当前topic没有queue的count信息，则直接从broker上获取queue的count信息。然后再缓存起来，在发送当前消息。然后第二次发送时，因为缓存里已经有了该消息，所以就不必再从broker拿了，且后续定时线程也会自动去更新该topic下的queue的count了。好，producer有了topic的queue的count，那用户在发送消息时，框架就能把这个topic的queueCount传递给用户，然后用户就能根据自己的需要将消息路由到第几个queue了。

4.consumer负载均衡如何实现

consumer负载均衡的意思是指，在消费者集群消费的情况下，如何让同一个consumer group里的消费者平均消费同一个topic下的queue。所以这个负载均衡本质上是一个将queue平均分配给consumer的过程。那么怎么实现呢？通过上面负载均衡的定义，我们只要，要做负载均衡，必须要确定consumer group和topic；然后拿到consumer group下的所有consumer，以及topic下的所有queue；然后对于当前的consumer，就能计算出来当前consumer应该被分配到哪些queue了。我们可以通过如下的函数来得到当前的consumer应该被分配到哪几个queue。



```
public class AverageAllocateMessageQueueStrategy : IAllocateMessageQueueStrategy
{
    public IEnumerable<MessageQueue> Allocate(string currentConsumerId,
IList<MessageQueue> totalMessageQueues, IList<string> totalConsumerIds)
    {
        var result = new List<MessageQueue>();

        if (!totalConsumerIds.Contains(currentConsumerId))
```

```
{
    return result;
}

var index = totalConsumerIds.IndexOf(currentConsumerId);
var totalMessageQueueCount = totalMessageQueues.Count();
var totalConsumerCount = totalConsumerIds.Count();
var mod = totalMessageQueues.Count() % totalConsumerCount;
var size = mod > 0 && index < mod ? totalMessageQueueCount / totalConsumerCount
+ 1 : totalMessageQueueCount / totalConsumerCount;
var averageSize = totalMessageQueueCount <= totalConsumerCount ? 1 : size;
var startIndex = (mod > 0 && index < mod) ? index * averageSize : index *
averageSize + mod;
var range = Math.Min(averageSize, totalMessageQueueCount - startIndex);

for (var i = 0; i < range; i++)
{
    result.Add(totalMessageQueues[(startIndex + i) % totalMessageQueueCount]);
}

return result;
}
```



函数里的实现就不多分析了。这个函数的目的就是根据给定的输入，返回当前consumer该分配到的queue。分配的原则就是平均分配。好了，有了这个函数，我们就能很方便的实现负载均衡了。我们可以对每一个正在运行的consumer内部开一个定时job，该job每隔一段时间进行一次负载均衡，也就是执行一次上面的函数，得到当前consumer该绑定的最新queue。因为每个consumer都有一个groupName属性，用于表示当前consumer属于哪个group。所以，我们就可以在负载均衡时到broker获取当前group下的所有consumer；另一方面，因为每个consumer都知道它自己订阅了哪些topic，所以有了topic信息，就能获取topic下的所有queue的信息了，有了这两样信息，每个consumer就能自己做负载均衡了。先看一下下面的代码：

```
_scheduleService.ScheduleTask(Rebalance, Setting.RebalanceInterval,
Setting.RebalanceInterval);
_scheduleService.ScheduleTask(UpdateAllTopicQueues,
Setting.UpdateTopicQueueCountInterval, Setting.UpdateTopicQueueCountInterval);
_scheduleService.ScheduleTask(SendHeartbeat, Setting.HeartbeatBrokerInterval,
Setting.HeartbeatBrokerInterval);
```

每个consumer内部都会启动三个定时的task，第一个task表示要定时做一次负载均衡；第二个task表示要定时更新当前consumer订阅的所有topic的queueCount信息，并把最新的queueCount信息都保存在本地；第三个task表示当前consumer会向broker定时发送心跳，这样broker就能通过心跳知道某个consumer是否还活着，broker上维护了所有的consumer信息。一旦有新增或者发现没有及时发送心跳过来的consumer，就会认为有新增或者死掉的consumer。因为broker上维护了所有的consumer信息，所以他就能提供查询服务，比如根据某个consumer group查询该group下的consumer。

通过这三个定时任务，就能完成消费者的负载均衡了。先看一下Rebalance方法：

```
private void Rebalance()
{
    foreach (var subscriptionTopic in _subscriptionTopics)
    {
        try
        {
            RebalanceClustering(subscriptionTopic);
        }
        catch (Exception ex)
        {
            _logger.Error(string.Format("[{0}]: rebalanceClustering for topic [{1}] has exception", Id, subscriptionTopic), ex);
        }
    }
}
```



代码很简单，就是对每个订阅的topic做负载均衡处理。再看一下RebalanceClustering方法：

[View Code](#)

上面的代码不多分析了，就是先根据consumer group和topic获取所有的consumer，然后对

consumer做排序处理。之所以要做排序处理是为了确保负载均衡时对已有的分配情况尽量不发生改变。接下来就是从本地获取**topic**下的所有**queue**，同样根据**queueId**做一下排序。然后就是调用上面的分配算法计算出当前**consumer**应该分配到哪些**queue**。最后调用**UpdatePullRequestDict**方法，用来对新增或删除的**queue**做处理。对于新增的**queue**，要创建一个独立的**worker**线程，开始从**broker**拉取消息；对于删除的**queue**，要停止其对应的**work**，停止拉取消息。

通过上面的介绍和分析，我们大家知道了**equeue**是如何实现消费者的负载均衡的。我们可以看出，因为每个**topic**下的**queue**的更新是异步的定时的，且负载均衡本身也是定时的，且**broker**上维护的**consumer**的信息也不是事实的，因为每个**consumer**发送心跳到**broker**不是实时发送的，而是比如每隔**5s**发送一次。所有这些因为都是异步的设计，所以可能会导致在负载均衡的过程中，同一个**queue**可能会被两个消费者同时消费。这个就是所谓的，我们只能做到一个消息至少被消费一次，但**equeue**层面做不到一个消息只会被消费一次。实际上像**rocketmq**这种也是这样的思路，放弃一个消息只会被消费一次的实现（因为代价太大，且过于复杂，实际上对于分布式的环境，不太可能做到一个消息只会被消费一次），而是采用确保一个消息至少会被消费一次（即**at least once**）。所以使用**equeue**，应用方要自己做好对每个消息的幂等处理。

5. 如何实现实时消息推送

消息的实时推送，一般有两种做法：推模式（**push**）和拉模式（**pull**）。**push**的方式是指**broker**主动对所有订阅了该**topic**的消费者推送消息；**pull**的方式是指消费者主动到**broker**上拉取消息；对于推模式，最大的好处就是实时，因为一有新的消息，就会立即推送给消费者。但是有一个缺点就是如果消费者来不及消费，它也会给消费者推消息，这样就会导致消费者端的消息会堵塞。而通过拉的方式，有两种实现：1）轮训的方式拉，比如每隔**5s**轮训一下是否有新消息，这种方式的缺点是消息不实时，但是消费进度完全由消费者自己把控了；2）开长连接的方式来拉，就是不轮训，消费者和**broker**之间一直保持的连接通道，然后**broker**一有新消息，就会利用这个通道把消息发送给消费者。

equeue中目前采用的是通过长连接拉取消息的方式。长连接通过**socket**长连接实现。但是虽然叫长连接，也不是一直不断开，而是也会设计一个超时的限制，比如一个长连接最大不超过**15s**，超过**15s**，则**broker**发送回复给**consumer**，告诉**consumer**当前没有新消息；然后**consumer**接收到这个回复后，就知道要继续发起下一个长连接来拉取。然后假如在这**15s**之内，**broker**上有新消息了，则**broker**就能立即主动利用这个长连接通知相应的消费者，把消息传给消费者。所以，可以看出，**broker**上在处理消费者的拉取消息的请求时，如果当前没有新消息，则会**hold**住这个**socket**连接，最多**hold 15s**，超过**15s**，则发送返回信息，告诉消费者当前无消息，然后消费者再次发送**pull message request**过来。通过这样的基于长连接的拉取模式，我们可以实现两个好处：1）消息实时推送；2）由消费者控制消息消费进度；

另外，**equeue**里还实现了消费者自身的自动限流功能。就是假如当前**broker**上消息很多，即生产者生产消息的速度大于消费者消费消息的速度，那**broker**上就会有消息被堆积。那此时消费者在拉取消息时，总是会有新消息拉取到，但是消费者又来不及处理这么多消息。所以**equeue**框架内置了一个限流（流控，流量控制）的设计，就是可以允许用于配制一个消费者端堆积的消息的上限，比如**3000**，超过这个数目（可配置），则**equeue**会让消费者以慢一点的频率拉取消息。比如延迟多少个毫秒（延迟时间可配置）再拉取。这样就简单的实现了流控的目的。

6. 如何处理消息消费失败的情况

作为一个消息队列，消费者总是可能会在消费消息时抛出异常，在**equeue**中这种情况就是消息消费失败的情况。通过上面的消费进度的介绍，大家知道了每个**queue**对某个特定的**consumer group**，都有一个唯一的消费进度。实际上，消息被拉取到**consumer**本地后，可能会被以两种方式消费，一种是并行消费，一种是线性消费。

并行消费的意思是，假如当前一次性拉取过来**32**个消息，那**equeue**会通过启动**task**（即开多线程）的方式并行消费每个消息；

线性消费的意思是，消息是在一个独立的单线程中顺序消费，消费顺序和拉取过来的顺序相同。

对于线性消费，假如前一个消息消费的时候失败了，也就是抛异常了，那该怎么办呢？可能想到的办法是重试个**3**次，但是要是重试后还是失败呢？总不能因为这个消息而导致后面的消息无法把消费吧？呵呵！对于这种情况，先说一下**rocketmq**里的处理方式吧：它的做法是，当遇到消费失败的情况，没有立马重试，而是直接把这个消息发送到**broker**上的某个重试队列，发送成功后，就可以往下消费下一个消息了。因为一旦发送到重试队列，那意味着这个消息就最后总是会被消费了，因为该消息不会丢了。但是要是发送到**broker**的重试队列也不成功呢？这个？！其实这种情况不大应该出现，如果出现，那基本就是**broker**挂了，呵呵。

rocketmq中，对于这种情况，那会把这个失败的消息放入本地内存队列，慢慢消费它。然后继续往后消费后面的消息。现在你一定很关心**queue**的**offset**是如何更新的？这里涉及到一个滑动门的概念。当一批消息从**broker**拉取到消费者本地后，并不是马上消费的，而是先放入一个本地的**SortedDictionary**，**key**就是消息在**queue**里的位置，**value**就是消息本身。因为是一个排序的**dictionary**，所以**key**最小的消息意味着是最前面的消息，最大的消息就是最后面的消息。然后不管是并行消费还是线性消费，只要某个消息被消费了，那就从这个**SortedDictionary**里移除掉。每次被移除一个消息时，总是会返回当前这个**SortedDictionary**里的最小的**key**，然后我们就能判断这个**key**是否和上次比是否前移了，如果是，则

更新queue的这个最新的offset。因为每次移除一个消息的时候，总是返回当前SortedDictionary里的最小的key，所以，假如当前offset是3，然后offset为4的这个消息一直消费失败，所以不会被移除，但是offset为5,6,7,8的这些消息虽然都消费成功了，但是只要offset为4的这个消息没有被移除，那最小的key就不会往前移动。这个就是所谓的滑动门的概念了。就好比是在铁轨上一辆在跑的动车，offset的往前移动就好比是动车在不断往前移动。因为我们希望offset总是会不断往前移动，所以不希望前面的某个消费失败的消息让这个滑动门停止移动（即我们总是希望这个最小的key能不断变大），所以我们会想方设法让消费失败的消息能不阻碍滑动门的往前移动。所以才把消费失败的消息放入重试队列。

另外一点需要注意一下：并不是每次成功消费完一个消息，就会立马告诉broker更新offset，因为这样那性能肯定很低，broker也会忙死，更好的办法是先只是在本地内存更新queue的offset，然后定时比如5s一次，将最新的offset更新到broker。所以，因为这个异步的存在，同样也会导致某个消息被重复消费的可能性，因为broker上的offset肯定比实际的消费进度要慢，有5s的时间差。所以，再次强调，应用方必须要处理好对消息的幂等处理！比如enode框架中，对每个command消息，框架内部都做了command的幂等处理。所以使用enode框架的应用，自身无需对command做幂等处理方面的考虑。

上面提到了并行消费和线性消费，其实对于offset的更新来说是一样的，因为并行消费无非是多线程同时从SortedDictionary中移除消费成功的消息，而单线程只是单个线程去移除SortedDictionary中的消息。所以我们要通过锁的机制，保证对SortedDictionary的操作是线程安全的。目前用了ReaderWriterLockSlim来实现对方法调用的线程安全。有兴趣的朋友可以去看一下代码。

最后，也是重点，呵呵。equeue目前还没有实现将失败的消息发回到broker的重试队列。这个功能以后会考虑加进去。

7.如何解决Broker的单点问题

这个问题比较复杂，目前equeue不支持broker的master-salve或master-master，而是单点的。我觉得一个成熟的消息队列，为了确保在一个broker挂了的时候，要尽量能确保有其他broker可以接替它，这样才能让消息队列服务器的可靠性。但是这个问题实在太复杂。rocketmq目前实现的也只是master-slave的方式。也就是只要主的master挂了，那producer就无法向broker发送消息了，因为slave的broker是只读的，不能直接接受新消息，slave的broker只能允许被consumer拉取消息。

这个问题，要讨论清楚，需要很多分布式方面的知识。由于篇幅的原因，这里就不做讨论了，实际上我自己也搞不清楚到底该如何设计。希望大牛们多多指点，如何实现broker的高可用哈！

分类： EQueue

好文要顶

关注我

收藏该文

netfocus

关注 - 21

粉丝 - 2111

荣誉：推荐博客

+加关注

800

« 上一篇： ENode 2.0 - 整体架构介绍

» 下一篇： 阿里巴巴-OS事业群-OS手机事业部-系统服务部门招聘Java开发工程师，有意者请进来

posted @ 2014-03-12 00:07 netfocus 阅读(31459) 评论(85) 编辑 收藏

< Prev

1

2

评论列表

#51楼 2015-07-07 15:44 rainrcn

@ netfocus

引用

@rainrcn

这个不是消息队列应该做的事情。你可以通过轮训的方式调用处理该消息的消费者所提供的服务来获取消息处理的结果。

谢谢你的回复，那你ENODE里是怎么实现发送一个消息而得到执行结果的呢。

支持(0) 反对(0)

#52楼[楼主] 2015-07-07 18:57 netfocus

http://www.cnblogs.com/netfocus/p/3595410.html

7/12

@ rainrcn
ENode中我是特定的场景的，特定方式处理，不是通用的。

支持(0) 反对(0)

#53楼 2015-08-25 10:23 xvipsservice

```
INSERT INTO [QueueOffset]
( Version ,
  ConsumerGroup ,
  Topic ,
  QueueId ,
  QueueOffset ,
  Timestamp
)
VALUES ( 1 ,
  'CommandConsumerGroup' ,
  'PostCommandTopic' ,
  0 ,
  0 ,
  '2015/8/25 星期二 10:03:55'
)
在win10 中文环境先，得到的DateTime.Now 为如上格式，导致报错。
```

支持(0) 反对(0)

#54楼 2015-09-16 09:44 朱立军

楼主，请问queue的消息什么时候发送给consumer，会有一个线程专门维护这个消息推送过程吗？

支持(0) 反对(0)

#55楼[楼主] 2015-09-16 10:04 netfocus

@ 朱立军
EQueue采用的是consumer主动到broker拉消息的模型，consumer长轮训到broker拉取消息。并不是broker推消息到consumer的。

支持(0) 反对(0)

#56楼 2015-09-16 10:49 朱立军

@ netfocus
consumer长轮询，不断想broker请求获取消息
但是可不可以这样了，consumer的长轮询操作依然保留，但是减少轮询的时间间隔，
另外producer发送消息至broker时做一次实时的推送（这里需要consumer发起的请求为异步，让服务器hold），这样做消息的推送可能会更实时，而且可以减少轮询的请求，但是唯一不好之处就hold住请求需要占用资源

支持(0) 反对(0)

#57楼[楼主] 2015-09-16 19:07 netfocus

@ 朱立军
consumer的pull request如果没有消息，则会被broker Hold住的呀，否则怎么叫长轮训呢？目前是hold 60s；如果这60s内还是没有新消息，那就返回给consumer，然后consumer再发送pull request。

broker会定时每隔1s扫描一下每个pull request是否有新的消息可用。也就是说，实时性最多延迟1s；另外，broker还支持配置，就是当消息到达时，立即同时相关的pull request。但这种通知如果在producer发送消息快而consumer消费消息慢的时候，是没必要的，因为consumer发起的所有pull request总是会立即拉取到消息的。只有producer的发送比consumer的消费频率慢时，才需要主动通知的。

你明白我的意思的吧？

基于以上的设计，完全不用担心消息的实时性。

支持(0) 反对(0)

#58楼 2015-09-18 09:29 朱立军

@ netfocus
明白了你的意思
但是我现在还有两个问题需要请教
（1）你这里hold pull request，是通过将这个request做同步处理，用一个线程来循环60，还是使用异步请求挂起？
（2）当消费速度大于生产速度时，就会出现produce的很多请求是无效，这对于资源是一种浪费，不知楼主关于这点有没有什么好的想法？

支持(0) 反对(0)

#59楼[楼主] 2015-09-18 12:38 netfocus

@ 朱立军
1) hold pull request的意思很简单，就是这一次socket请求，暂时先不发送回复即可。等到60s后，再发送回复，如何做
到60s后发送回复？先把所有hold住的request都放在一个dict里，然后用一个独立的线程定时扫描哪些到了60s即可；
定时线程每秒扫描一次。

2) 消费速度大于生产速度时，怎么会有大量的无效请求呢？假如有10个consumer，那这10个consumer每个也就每60
s才发送一次pull request呀。因为pull request还没回来之前，是不会发下一个pull request的。不然怎么叫长轮训呢。
支持(0) 反对(0)

#60楼 2015-09-18 14:39 朱立军

@ netfocus
Hold pull request，hold住60s后再回复，那么发起这次请求的客户端岂不是要等待60s？
支持(0) 反对(0)

#61楼[楼主] 2015-09-18 15:13 netfocus

@ 朱立军
请求都是异步发送的，发送后并没有同步等待结果。
支持(0) 反对(0)

#62楼 2015-09-18 15:16 朱立军

第一次发起异步请求，查看有没有消息，有则返回响应，并重新发起请求，
没有消息，则挂起等待，直到有消息退过，然后消息主动的匹配请求，调用请求的回调函数处理
请问是这么个逻辑吗？
支持(0) 反对(0)

#63楼[楼主] 2015-09-18 18:43 netfocus

@ 朱立军
请求是异步发送的，如果没有消息，也不会挂起当前发送消息的线程。你要先理解异步Socket通信。
支持(0) 反对(0)

#64楼 2015-09-22 09:03 朱立军

@ netfocus
您好，我这边不是采用的Socket通信，我Web端的通信，客户端异步发送一个http请求，服务端也做异步处理，挂起这
个请求，待有消息就用回调函数进行请求处理
那么，我想问下，这么做性能有什么瓶颈之处不？
支持(0) 反对(0)

#65楼[楼主] 2015-09-22 10:16 netfocus

你直接加我的QQ群咨询吧。这样交流太累了。185916873
支持(0) 反对(0)

#66楼 2015-10-13 14:42 472790378

@汤兄
在知道你的消息队列前，我做了一个面向业务的消息队列，请给予建议。
后来了解你做的消息队列后，我也开源了。
开源的.net业务消息队列
博文:<http://my.oschina.net/u/2379842/blog/515860>
项目:<http://git.oschina.net/chejiangyi/Dyd.BusinessMQ>
支持(0) 反对(0)

#67楼 2015-10-13 14:43 472790378

。net消息队列
支持(0) 反对(0)

- #68楼 2015-10-20 10:45 ...后知后觉

根据博主的论述，Queue对于每一个Consumer Group，其offset都是唯一的，故而不论Consumer 是否同时消费同一个队列，其应该都可以保证不会重复消费。博主不能保证的原因应该这是由于批量拉取和延迟更新Queue Offset的原因。关于滑动门的解释好像并不是很清楚。

个人由于业务原因也正在研究消息队列，有空希望和博主可以多多交流。

QQ: 582105721

支持(0) 反对(0)
- #69楼 2015-11-06 13:19 牛！

楼主，是否可以单独设置每一个Topic的消费模式，如何设置一个Topic为集群消费模式或广播消费模式？

支持(0) 反对(0)
- #70楼[楼主] 2015-11-06 16:44 netfocus

@ 牛！
不支持哦。

支持(0) 反对(0)
- #71楼 2015-11-06 17:10 牛！

@ netfocus
@netfocus
引用
@牛！
不支持哦。

如何设置一个Topic为集群消费模式或广播消费模式？

支持(0) 反对(0)
- #72楼[楼主] 2015-11-06 21:12 netfocus

不支持配置的，只有集群消费。

支持(0) 反对(0)
- #73楼 2015-12-02 09:25 Realm_King

楼主，ECommon基础类在哪里呢

支持(0) 反对(0)
- #74楼 2016-01-20 11:23 piyongcai

首先谢谢你的成果及无私奉献，同时提出如下建议：

1、消息传输部分可参看NetMQ或者NanoMsg，轻量化设计，性能超高。支持多种消息模式。

2、消息存储部分，可以考虑topic分片，每个分片单独一个日志文件。

3、ECommon类库中JSON序列化，建议更换为JIL（支持匿名对象序列化），如果需要序列化的对象都不是匿名对象，可以考虑使用NetJSON，性能超高。我有详细对比测试数据。

4、更高层次封装，隐藏调用细节。增加远程调用代理类，实现远程调用。比如生成者需要计算 a+b，消费者计算结果并返回，生产者可以通过调用函数函数调用。

var sum = BrokerAgent.CalcAdd(a,b);

5、内置消息压缩功能。

6、消息存储增加开关配置，某些应用容许消息丢失，但是会追求最大的并发量。比如OLTP类业务应用。

7、内置加密，实现传输安全。支持多种加密或者签名，比如ECC、3DES等。

再次谢谢你的成果及无私奉献。

另外加入QQ群185916873，答案是多少呀？我的QQ是7639621

支持(0) 反对(0)
- #75楼[楼主] 2016-01-20 13:06 netfocus

@ piyongcai
感谢你的回复，有些点我会考虑实现一下。

但我也有些不同的看法：

1.消息传输，通信层我会使用自己实现的ECommon中的TCP通信层，性能一样高；

- 2.消息存储，我不会考虑topic分片，一个分片一个日志文件；我设计equeue参考的是阿里的rocketmq；所有的消息都写入一个日志文件，这样一台broker才可以支持更多的topic；
- 3.序列化，目前使用的是Newtonsoft.Json，虽然其性能不是最高，但功能最强大；你推荐的我去学习下，可以考虑使用；
- 4.远程RPC调用，MQ不需要支持这种特性；MQ只需要实现pub-sub通信方式即可。
- 5.内置消息压缩功能、内存消息模式、内置加密，这3个确实可以支持；

支持(1) 反对(0)

#76楼 2016-01-21 08:56 虾。

@ piyongcai
Topic分片是什么？

支持(0) 反对(0)

#77楼 2016-01-21 09:04 虾。

@ piyongcai
明白了

支持(0) 反对(0)

#78楼 2016-02-15 11:39 euler

请问，broker消息持久化使用的是什么nosql工具？

支持(0) 反对(0)

#79楼 2016-05-18 17:01 SnowX

学习

支持(0) 反对(0)

#80楼 2016-05-22 21:28 王庆东mas

因为broker上topic的queue的数量一般不会变化，所以这样的缓存很有意义。

请问大神，为啥queue的数量一般不会变化呢？

支持(0) 反对(0)

#81楼 2016-05-22 21:31 王庆东mas

好，producer有了topic的queue的count，那用户在发送消息时，框架就能把这个topic的queueCount传递给用户，然后用户就能根据自己的需要将消息路由到第几个queue了。

假如queueCount数量为10，那么用户只知道10这个数字，具体怎么知道这10个哪个是自己想要的目标queue呢？怎么知道消息路由到第几个queue了。

支持(0) 反对(0)

#82楼 2016-05-22 21:42 王庆东mas

信息量很大，受益匪浅！谢谢汤神！

支持(0) 反对(0)

#83楼 2016-05-22 21:48 王庆东mas

broker的重试队列 有相关文章吗？

支持(0) 反对(0)

#84楼[楼主] 2016-05-24 10:11 netfocus

@ Ong_Ching_Tong
引用

好，producer有了topic的queue的count，那用户在发送消息时，框架就能把这个topic的queueCount传递给用户，然后用户就能根据自己的需要将消息路由到第几个queue了。

假如queueCount数量为10，那么用户只知道10这个数字，具体怎么知道这10个哪个是自己想要的目标queue呢？怎

么知道消息路由到第几个queue了。

producer会获取到所有的queue的ID

支持(1) 反对(0)

#85楼 2016-09-25 16:40 lyt208

我如果不想用代码中的生产者Producer而是想自己做一个生产者，即通过Socket接收到数据再转发到broker，怎么用啊！

支持(0) 反对(0)

刷新评论 刷新页面 返回顶部

注册用户登录后才能发表评论，请 [登录](#) 或 [注册](#)，[访问](#)网站首页。

- 【推荐】50万行VC++源码：大型组态工控、电力仿真CAD与GIS源码库
- 【活动】优达学城正式发布“无人驾驶车工程师”课程
- 【推荐】移动直播百强八成都在用融云即时通讯云
- 【推荐】别再闷头写代码！找对工具，事半功倍，全能开发工具包用起来
- 【福利】网易云信1周年接入开发者突破10万，送红包活动火热开展中



快速开发WEBUI

提升开发效率、降低开发成本、加速项目进度。
下载试用

miniui.com



广告

最新IT新闻：

- 太拼了 中兴手机最早创始人之一时隔14年出任公司CEO
- 为什么我不做VC了？
- 百度地图发布鹰眼硬件联盟 首批接入联发科
- 95后喜欢什么样儿的互联网公司？
- 豆瓣影业这艘“飞船”，能否真正腾空而起？

» 更多新闻...

极光 智能推送全面升级 更快、更稳定、更成熟 [了解更多](#)

最新知识库文章：

- 技术的正宗与野路子
- 陈皓：什么是工程师文化？
- 没那么难，谈CSS的设计模式
- 程序猿媳妇儿注意事项
- 可是姑娘，你为什么要编程呢？

» 更多知识库文章...

历史上的今天：

2013-03-12 分布式系统英文参考资料