

05 | 消息序号生成器：如何保证你的消息不会乱序？

2019-09-06 袁武林

即时消息技术剖析与实战

[进入课程 >](#)



讲述：袁武林

时长 16:49 大小 23.12M



你好，我是袁武林。

前面几节课，我们较为系统地介绍了如何解决消息实时到达的问题，也对如何保证消息可靠投递的几种实战中常用的方式进行了一一讲解。

那么，今天的课程我们继续一起聊一聊，IM 系统设计中另一个比较复杂，但又非常重要的话题：消息收发的一致性。需要提醒的是，我们这里的讲到的一致性，一般来说是指消息的时序一致性。

为什么消息的时序一致性很重要？

对于聊天、直播互动等业务来说，消息的时序代表的是发送方的意见表述和接收方的语义逻辑理解，如果时序一致性不能保证，可能就会造成聊天语义不连贯、容易出现曲解和误会。

你可以想象一下，一个人说话颠三倒四，前言不搭后语的样子，就理解我们为什么要尤其注重消息的时序一致性了。

对于点对点的聊天场景，时序一致性需要保证接收方的接收顺序和发送方的发出顺序一致；而对于群组聊天，时序一致性保证的是群里所有接收人看到的消息展现顺序都一样。

为什么保证消息的时序一致性很困难？

从理论上来说，保持消息的时序一致性貌似并不难。理论上，我们想象的消息收发场景中，只有单一的发送方、单一的接收方。

如果发送方和接收方的消息收发都是单线程操作，并且和 IM 服务端都只有唯一的一个 TCP 连接，来进行消息传输，IM 服务端也只有一个线程来处理消息接收和消息推送。这种场景下，消息的时序一致性是比较容易能得到保障的。

但在实际的后端工程实现上，由于单发送方、单接收方、单处理线程的模型吞吐量和效率都太低，基本上不太可能存在。

更多的场景下，我们可能需要面对的是多发送方、多接收方、服务端多线程并发处理的情况。所以，知道了难点，我们再来看一看究竟在后端的工程实现上，保证消息的时序一致都存在哪些难点。

消息的时序一致性其实是要求我们的消息具备“时序可比较性”，也就是消息相对某一个共同的“时序基准”可以来进行比较，**所以，要保证消息的时序一致性的一个关键问题是：我们是否能找到这么一个时序基准，使得我们的消息具备“时序可比较性”。**

在工程实现上，我们可以分成这样几步。

首先是：如何找到时序基准。

其次是：时序基准的可用性是不是可以。

最后是：有了时序基准，还有其他的误差吗，有什么办法可以减少这些误差？

如何找到时序基准？

下面我从消息收发的实际场景来分析一下，收发过程中如何找到一个全局的“时序基准”。在这里，我们来逐一分析一下。

首先，我们来看看发送方的本地序号和本地时钟是否可以作为“时序基准”？

这里解释一下，所谓发送方的本地序号和本地时钟是指发送方在发送消息时，连同消息再携带一个本地的时间戳，或者本地维护的一个序号给到 IM 服务端。

IM 服务端再把这个时间戳或者序号和消息，一起发送给消息接收方，消息接收方根据这个时间戳或者序号，来进行消息的排序。

仔细分析一下，貌似发送方的本地序号或者本地时钟不适合用来作为接收方排序的“时序基准”，原因有下面几点。

1. 发送方时钟存在较大不稳定因素，用户可以随时调整时钟导致序号回退等问题。
2. 发送方本地序号如果重装应用会导致序号清零，也会导致序号回退的问题。
3. 类似“群聊消息”和“单用户的多点登陆”这种多发送方场景，都存在：同一时钟的某一时间点，都可能有多条消息发给同一接收对象。比如同一个群里，多个人同时发言；或者同一个用户登录两台设备，两台设备同时给某一接收方发消息。多设备间由于存在时钟不同步的问题，并不能保证设备带上来的时间是准确的，可能存在群里的用户 A 先发言，B 后发言，但由于用户 A 的手机时钟比用户 B 的慢了半分钟，如果以这个时间作为“时序基准”来进行排序，可能反而导致用户 A 的发言被认为是晚于用户 B 的。

因此以发送方的本地时钟或者本地序号作为“时序基准”是不可靠的。**那么，我们接下来看看 IM 服务器的本地时钟是否可以作为“时序基准”？**

这里也解释一下，IM 服务器的本地时钟作为“时序基准”是指：发送方把消息提交给 IM 服务器后，IM 服务器依据自身服务器的时钟生成一个时间戳，再把消息推送给接收方时携带这个时间戳，接收方依据这个时间戳来进行消息的排序。

我们分析一下，好像 IM 服务器的本地时钟作为接收方消息排序的“时序基准”也不太合适。

这是因为，在实际工程中，IM 服务都是集群化部署，集群化部署也就是许多服务器同时部署任务。

虽然多台服务器通过 NTP 时间同步服务，能降低服务集群机器间的时钟差异到毫秒级别，但仍然还是存在一定的时钟误差。

而且 IM 服务器规模相对比较大，时钟的统一性维护上也比较有挑战，整体时钟很难保持极低误差，因此一般也不能用 IM 服务器的本地时钟来作为消息的“时序基准”。

既然单机本地化的时钟或者序号都存在问题，那么如果有一个全局的时钟或者序号，是不是就能解决这个问题了呢？我们来看看 IM 服务端如果有一个全局序号，是不是就可以作为“时序基准”。

如果所有的消息的排序都依托于这个全局的序号，这样就不存在时钟不同步的问题了。

比如说如果有一个全局递增的序号生成器，应该就能避免多服务器时钟不同步的问题了，IM 服务端就能通过这个序号生成器发出的序号，来作为消息排序的“时序基准”。

这种“全局序号生成器”可以通过多种方式来实现，常见的比如 Redis 的原子自增命令 incr，DB 自带的自增 ID，或者类似 Twitter 的 snowflake 算法、“时间相关”的分布式序号生成服务等。

以上，我们了解到 IM 服务端可以用全局序号生成器，来做为时序基准。

如何确保“时序基准”的可用性

使用“全局序号生成器”发出的序号，来作为消息排序的“时序基准”，能解决每一条消息没有标准“生产日期”的问题，但如果是面向高并发和需要保证高可用的场景，还需要考虑这个“全局序号生成器”的可用性问题。

首先，类似 Redis 的原子自增和 DB 的自增 ID，都要求在主库上来执行“取号”操作，而主库基本都是单点部署，在可用性上的保障会相对较差，另外，针对高并发的取号操作这个单点的主库可能容易出现性能瓶颈。

而采用类似 snowflake 算法的时间相关的分布式“序号生成器”也存在一些问题。

一个是发出的号携带的时间精度有限，一般只能到秒级或者毫秒级，比如微博的 ID 生成器就是精确到秒级的。另外由于这种服务大多都是集群化部署，携带的时间采用的服务器时间，也存在时钟不一致的问题（虽然时钟同步上比控制大量的 IM 服务器也相对容易一些）。

由上可知，基于“全局序号生成器”仍然存在不少问题，那这样是不是说基于“全局序号生成器”生成的序号来对消息进行排序的做法不可行呢？

我们从后端业务实现的角度，来具体分析一下。

从业务层面考虑，对于群聊和多点登陆这种场景，没有必要保证全局的跨多个群的绝对时序性，只需要保证某一个群的消息有序即可。

这样的话，如果可以针对每一个群有独立一个“ID 生成器”，能通过哈希规则把压力分散到多个主库实例上，大量降低多群共用一个“ID 生成器”的并发压力。

对于大部分即时消息业务来说，产品层面可以接受消息时序上存在一定的细微误差。比如同一秒收到同一个群的多条消息，业务上是可以接受这一秒的多条消息，未严格按照“接收时的顺序”来排序的。实际上，这种细微误差对于用户来说，基本也是无感知的。

那么，对于依赖“分布式的时间相关的 ID 生成器”生成的序号来进行排序，如果时间精度业务上可以接受，也是没问题的。

从之前微信对外的分享，我们可以了解到：微信的聊天和朋友圈的消息时序也是通过一个“递增”的版本号服务来进行实现的。不过这个版本号是每个用户独立空间的，保证递增，不保证连续。

微博的消息箱则是依赖“分布式的时间相关的 ID 生成器”来对私信、群聊等业务进行排序，目前的精度能保证秒间有序。

如何解决“时序基准”之外的其他误差

有了“时序基准”，是不是就能确保消息能按照“既定顺序”到达接收方呢？答案是并不一定能做到。原因在于下面两点。

1. IM 服务器都是集群化部署，每台服务器的机器性能存在差异，因此处理效率有差别，并不能保证先到的消息一定可以先推送到接收方，比如有的服务器处理得慢，或者刚好碰到一次 GC，导致它接收的更早消息，反而比其他处理更快的机器更晚推送出去。
2. IM 服务端接收到发送方的消息后，之后相应的处理一般都是多线程进行处理的，比如“取序号”“暂存消息”“查询接收方连接信息”等，由于多线程处理流程，并不能保证先取到序号的消息能先到达接收方，这样的话对于多个接收方看到的消息顺序可能是不一致的。

所以，我们一般还需要端上能支持对消息的“本地整流”。我们来看一下，如何去做本地整流。

消息服务端包内整流

虽然大部分情况下，聊天、直播互动等即时消息业务能接受“小误差的消息乱序”，但某些特定场景下，可能需要 IM 服务能保证绝对的时序。

比如发送方的某一个行为同时触发了多条消息，而且这多条消息在业务层面需要严格按照触发的时序来投递。

一个例子：用户 A 给用户 B 发送最后一条分手消息同时勾上了“取关对方”的选项，这个时候可能会同时产生“发消息”和“取关”两条消息，如果服务端处理时，把“取关”这条信令消息先做了处理，就可能导致那条“发出的消息”由于“取关”了，发送失败的情况。

对于这种情况，我们一般可以调整实现方式，在发送方对多个请求进行业务层合并，多条消息合并成一条；也可以让发送方通过单发送线程和单 TCP 连接能保证两条消息有序到达。

但即使 IM 服务端接收时有序，由于多线程处理的原因，真正处理或者下推时还是可能出现时序错乱的问题，解决这种“需要保证多条消息绝对有序性”可以通过 IM 服务端包内整流来实现。

整个过程是这样的：

首先生产者为每个消息包生成一个 packageID，为包内的每条消息加个有序自增的 seqID；

其次消费者根据每条消息的 packageID 和 seqID 进行整流，最终执行模块只有在一定超时时间内完整有序地收到所有消息才执行最终操作，否则根据业务需要触发重试或者直接放弃操作。

服务端包内整流大概就是这个样子，我们要做的是在最终服务器取到 TCP 连接后下推的时候，根据包的 ID，对一定时间内的消息做一个整流和排序。

消息接收端整流

携带不同序号的消息到达接收端后，可能会出现“先产生的消息后到”“后产生的消息先到”等问题。消息接收端的整流就是解决这样的一个问题，

消息客户端本地整流的方式可以根据具体业务的特点来实现，目前业界比较常见的实现方式也很简单，步骤如下：

1. 下推消息时，连同消息和序号一起推送给接收方；
2. 接收方收到消息后进行判定，如果当前消息序号大于前一条消息的序号，就将当前消息追加在会话里；
3. 否则继续往前查找倒数第二条、第三条等，一直查找到恰好小于当前推送消息的那条消息，然后插入在其后展示。

小结

今天我们讲到了在多发送方、多接收方、服务端多线程并发处理的情况下，保持消息时序一致性的重要性以及处理方法。

对于聊天、直播互动等即时消息的场景，保持消息的时序一致性能避免发送方的意见表述和接收方的语义理解出现偏差的情况。

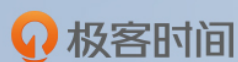
对于如何保持消息的时序一致性的关键点在于：需要找到一个时序基准来标识每一条消息的顺序。

这个时序基准可以通过全局的序号生成器来确定。常见的实现方式包括支持单调自增序号的资源生成，或者分布式时间相关的 ID 生成服务生成。两种方式各有一些限制，不过，你都可以根据业务自身的特征来进行选择。

有了通过时序基准确定的消息序号，由于 IM 服务器差异和多线程处理的方式，不能保证服务端的消息一定能按顺序推到接收方。我们可以通过“服务端包内整流”机制来保证需要“严格有序”批量消息的正确执行；或者，接收方根据消息序号来进行消息本地整流，从而确保多接收方的最终一致性。

最后给你留一道思考题。在即时消息收发场景中，用于保证消息接收时序的序号生成器为什么可以不是全局递增的？

以上就是今天课程的内容，欢迎你给我留言，我们可以在留言区一起讨论。感谢你的收听，我们下期再见。



即时消息技术剖析与实战

10 周精通 IM 后端架构技术点

袁武林

微博研发中心技术专家



新版升级：点击「 请朋友读」，20位好友免费读，邀请订阅更有**现金**奖励。

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 04 | ACK机制：如何保证消息的可靠投递？

下一篇 06 | HttpDNS和TLS：你的消息聊天真的安全吗？

精选留言 (23)



Alexdown

写留言



2019-09-06

希望老师多补充一些流程图或代码来帮助理解，光看文字因经验不足，有时很难想象理解，甚至会引发歧义。谢谢

展开 ▾

💬 2

👍 24



一个爱编程的胖子

2019-09-08

可不可以适当添加一些代码示例。单纯的文字描述比较干

作者回复: 加了两张流程图来说明一下服务端整流这一块的应用场景和过程，更新后再看能不能理解。



💬

👍 2



卫江

2019-09-06

问题：为什么不需要全局自增id？原因：因为没有全局排序的需求。而且全局自增id肯定有单点和性能问题。我们目前的需求有两点：单聊和群聊。单聊我们可以通过针对于会话id的自增id解决，群聊通过基于群id的自增id解决，这样就拥有了不错的扩展性，避免单点和性能问题，当然了如果群很大也许也有问题，同时这种方式也可以控制某个id生成服务出问题影响的范围。

展开 ▾

作者回复: 嗯，也考虑如果有“最近联系人列表”页的需求，需要按照多个群或者多个直播间的最新一条消息的产生先后来排序，这种情况可能还需要考虑使用其他属性来进行全局排序了（比如消息产生的时间戳）。



💬

👍 2




王棕生

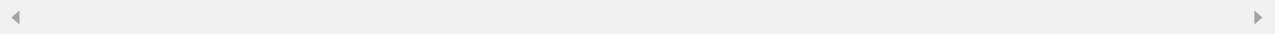
2019-09-06

问题：在即时消息收发场景中，用于保证消息接收时序的序号生成器为什么可以不是全局递增的？

答：这是由业务场景决定的，这个群的消息和另一个群的消息在逻辑上是完全隔离的，只要保证消息的序号在群这样的一个局部范围内是递增的即可；当然如果可以做到全局递增最好，但是会浪费很多的资源，却没有带来更多的收益。...

展开 ▾

作者回复: 



👍 1



小肚臍era

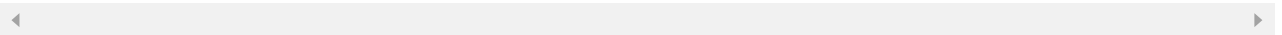
2019-09-06

对老师提的问题，提出一下自己的想法。

因为实时通信里是会存在多个会话的，如果用于保证消息接收时序的序号生成器是全局递增的，即用于保证所有会话的接收时序，那么会存在可用性和性能瓶颈问题。

其次，正如老师提到的，从业务层面考虑，对用户来说，只要保证单个会话里消息的时序是正确的即可，因为不同的会话相关性一般不强，不需要保证严格的时序同步，所以针...
展开 ▾

作者回复: 是这样的，会话内的排序只需要保证会话内序号递增就可以。当然，如果有“最近联系人列表”这种需求，还需要考虑跨多个会话进行排序的情况，这个可能需要其他属性来协助全局排序了。



👍 1



刘小兔bunny

2019-09-09

希望可以做一些演示类的代码，纯文字表述比较浮于表面



moooofly

2019-09-09

老师能够提供一个示意图，在途中标明 packageID, seqID 和 xxxID 等都用在什么位置，感觉看过文章和留言问答后，都搞不清楚哪些 ID 用哪里了，多谢~

展开 ▾



墙角儿的花

2019-09-08

多谢老师，受益良多。希望和老师多交流。

老师讲的防止业务执行错乱的整流方案，是通过package打包，类似逻辑集装箱，将包内消息有序处理，原理上很清晰，但落地实现比较困难，不太好掌握package的边界。究竟从哪个时机到哪个时机范围的消息归为一个包呢？也希望老师赐教个方法。

...

展开 ∨

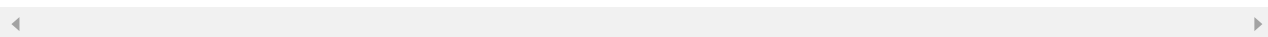
作者回复: 多多交流~

这里讲的服务端整流可以参考离线消息的推送，比如离线消息推送时，用户的多条消息需要推送，这多条消息在服务端进行多线程处理时可能出现乱序的情况，通过在取离线消息时，给每条消息使用同一个packageId并自增一个seq，那么网关机在最终推送时，就可以根据这个packageId来进行一次整流，保证最终下推时消息的有序。

方案1的思路没问题哈，考虑下很多IM场景，由于服务端一般是多层的架构，比如业务层，网关层，会涉及到多个进程的处理，中间的流转可能需要经过消息队列，这些过程也可能导致乱序出现。

方案2的话主要是通用性上可能不是太好，需要区分消息类型啥的，处理逻辑也稍微复杂一些。

实际上，大部分IM场景有了接收端的整流是不太需要服务端整流的，除了服务端可能存在短时间内推送多条连续消息的情况才可能需要服务端进行整流。



clip

2019-09-08

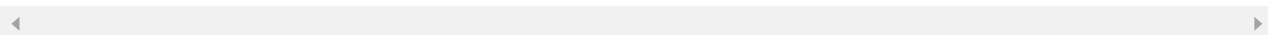
如果我们应用场景就是 IM，消息接收端整流是不是可能导致收到消息之后也能又往上面插入了新消息？感觉这种体验可能不是很好。

是不是要采取另外的措施？

能想到的有：必须保证消息顺序下发，如果中间有比较耗时的消息可能先用一个占位符代替。自己发送的消息要等服务端确认后才真正进入消息流排序。

展开 ∨

作者回复: 保证服务端推送是必须有序也是可以的，看具体的需求场景是否真的需要，另外实现成本也需要考虑。接收端整流也可以是如果乱序先不显示而是等一段时间，来尽量避免显示上的跳变。



clip

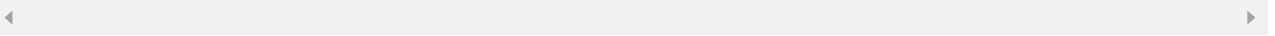
2019-09-08

服务端包内整流那块儿有点不理解。

这个 pkg 指的是类似举例的“最后一条消息和取关操作”那种打包在一起的事件吗？还是群纬度的那种一个群的算一个 pkg？

作者回复: 新补充了两个图, 等更新了大家可以看一下哈~

比如离线消息推送时, 用户的多条消息需要推送, 这多条消息在服务端进行多线程处理时可能出现乱序的情况, 通过在取离线消息时, 给每条消息使用同一个packageId并自增一个seq, 那么网关机在最终推送时, 就可以根据这个packageId来进行一次整流, 保证最终下推时消息的有序。



1



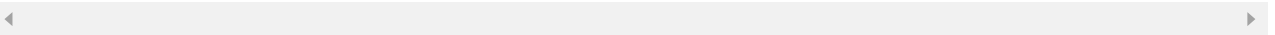
云师兄

2019-09-07

文中提到的packageid+seqid整流, 其中packageid就是根据业务场景设置, 可能是用户维度, 群聊维度吗

展开

作者回复: packageid可以理解为一次需要保证时序的多条消息和信令的集合。比如, 用户上线获取离线消息时需要推送多条消息, 这多条消息就可以是一个packageid和多个不同的seqid。



Alpha

2019-09-07

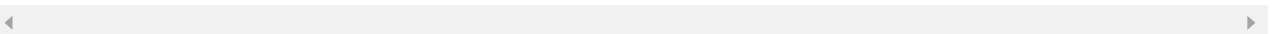
生产者每个消息包生成一个 packageID。

——请问这里的消息包是什么概念, 是指多个消息的集合吗? 如果是的话, 什么场景下会将多个消息作为一个包一起发送呢?

或者指的是包含 一条消息 + 一条指令 的集合吗?

展开

作者回复: 比如离线消息推送时, 用户的多条消息需要推送, 这多条消息在服务端进行多线程处理时可能出现乱序的情况, 通过在取离线消息时, 给每条消息使用同一个packageid并自增一个seq, 那么网关机在最终推送时, 就可以根据这个packageid来进行一次整流, 保证最终下推时消息的有序。



一路向北

2019-09-07

多台服务器, 如果都使用北京标准时间, 可不可以用服务端的时间戳呢?

作者回复: 主要是多台服务器的时钟同步问题, 集群越大差异性也越大。



小可

2019-09-07

只要能保证发送到同一个接收端的取到的序号是自增的，就不需要全局自增

老师在消息接收端整理的环节有点疑问：

由于网络或者其他原因导致消息到达接受端的序号是1、2、5、4，按照您提到的处理规则，展现到回话里顺序是1、2、4、5，如果后面再来个消息3，已经找不到比它序号小的消息了，该如何展现？

展开 ▾

作者回复: 这一篇讲的是时序性，时间戳或者序号大小的比较只解决时序问题，不能解决丢消息的问题。要保证消息不丢，序号不连续也是不行的。

💬 1



煜

2019-09-06

老师，我对您这句话感到疑惑，“最终执行模块只有在一定超时时间内完整有序地收到所有消息才执行最终操作”，服务端或者客户端收到一条消息，然后是等待后面来的消息进行排序呢还是直接就推送或显示呢？

展开 ▾

作者回复: 这个看需求，客户端接收到消息即使不连续一般也可以直接先显示，然后等前面的消息到了再在页面进行一次插入排序。也有的实现，如果接收到不连续的消息会尝试等待一个非常短的时间看前面的消息会不会到，如果还没到也一般会直接显示出来。



墙角儿的花

2019-09-06

老师，帮忙解答一下我的这篇文章另个留言提的问题，谢谢



Peiel

2019-09-06

老师您好，在讲包内整理的流程中，客户端本地整理的流程中，不需要使用服务端的 packageID 吗？只用到 seqID 来判断顺序就可以了么？

展开 ▾

作者回复: 服务端整流用到的这个seqID和客户端本地排序使用的seqID是不相关的哈, 服务端整流用到的这个seqID不会给到接收端。



阳仔

2019-09-06

snowflake 算法的时间相关的分布式“序号生成器”也存在一些问题?
这个是什么问题呢? 可以提一下吗

展开 ▾

作者回复: 主要是时间精度不太够, 只能做到秒间有序, 所以排序上可能会有一定的误差。



小祺

2019-09-06

感觉消息接收端整流是必做的, 那服务端的整流是不是就显得多余了?

作者回复: 如果只是消息推送的话, 接收端的整流基本就ok了, 但是通道里推送的不仅仅是消息, 还有信令(比如删除某一个会话的信令), 这种情况服务端整流能够减少消息和信令乱序推送到接收端后导致端上逻辑异常的问题。



墙角儿的花

2019-09-06

对于“分手”、“取关”的严格业务顺序场景, 通过单线程单tcp连接能保证消息一定按着发送顺序到达服务器吗? socket.send有序发送两条消息A 和 B, 由于链路故障是否可能导致服务器先接收到B后接收到A, 我一直保持着业务层消息即使同一tcp连接上有序发送也有可能出现乱序到达, 所以需要接收端在业务层重排的认知, 但好像也确实没有证明。

展开 ▾

作者回复: 单连接单线程的话TCP层的“有序接收”能够保证消息的有序到达。但这种模型的性能和可用性基本不能用在真实业务场景里。

