04 | ACK机制：如何保证消息的可靠投递?

2019-09-04 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 13:33　大小 12.42M

你好，我是袁武林。

在第一节的课程中，我们说到了即时消息系统中的四个重要特性，实时性、可靠性、一致性、安全性。

上一节课我们从如何保证消息实时性方面，了解了业界常用的一些方式以及背后具体的原理。那么今天我们接着来讲一讲，在即时消息的系统架构设计里，如何来保证消息的可靠投递。

首先，我们来了解一下，什么是消息的可靠投递?

站在使用者的角度来看，消息的可靠投递主要是指：消息在发送接收过程中，能够做到不丢消息、消息不重复两点。

这两个特性对于用户来讲都是非常影响体验的。我们先说一下不丢消息。

试想一下，你把辛辛苦苦攒到的零花钱打赏给了中意的“主播小姐姐”，但由于系统或者网络的问题，这条对你来说至关重要的打赏消息并没有成功投递给“主播小姐姐”，自然也就没有后续小姐姐和你一对一的互动环节了，想想是不是很悲剧？

消息重复也不用多说，谁也不愿意浪费时间在查看一遍又一遍的重复内容上。

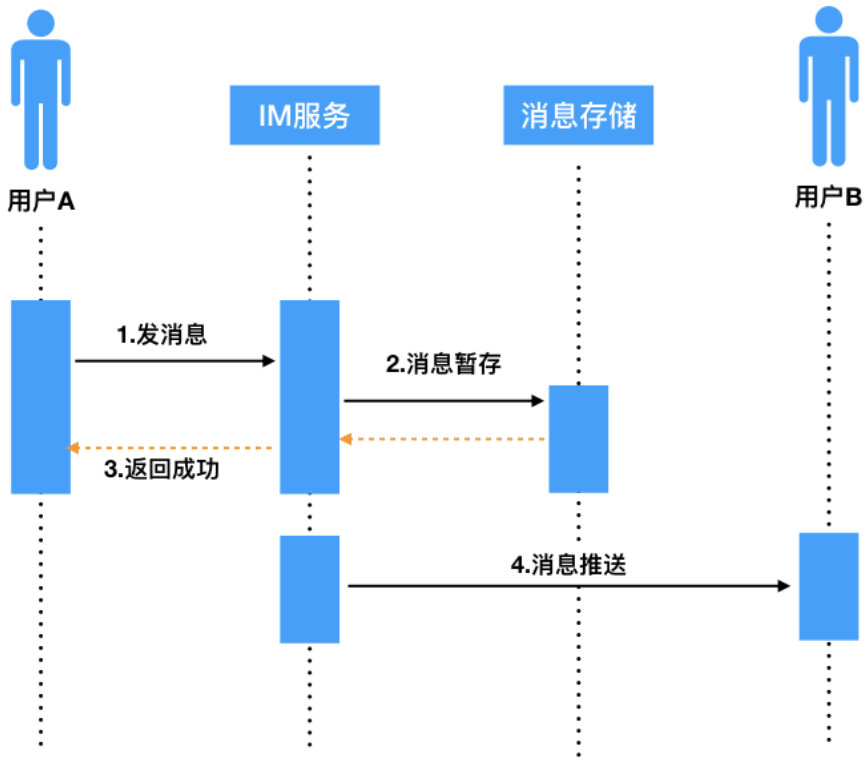
那么在一般的 IM 系统的设计中，究竟是如何解决这两大难题的呢？下面我们结合一些简单的案例，来看一看“不丢消息”“消息不重复”这些能力，在技术上到底是怎么实现的。

消息丢失有哪几种情况？

我们以最常见的“服务端路由中转”类型的 IM 系统为例（非 P2P），这里解释一下，所谓的“服务端路由中转”是指：一条消息从用户 A 发出后，需要先经过 IM 服务器来进行中转，然后再由 IM 服务器推送给用户 B，这个也是目前最常见的 IM 系统的消息分发类型。

我们可以把它和少数 P2P 类型区别一下，P2P 类型的消息投递是直接由用户 A 的网络发送到用户 B 的网络，不经过服务端路由。

那么，我们来假设一个场景：用户 A 给用户 B 发送一条消息。接下来我们看看哪些环节可能存在丢消息的风险？



参考上面时序图，发消息大概整体上分为两部分：

用户 A 发送消息到 IM 服务器，服务器将消息暂存，然后返回成功的结果给发送方 A（步骤 1、2、3）；

IM 服务器接着再将暂存的用户 A 发出的消息，推送给接收方用户 B（步骤 4）。

其中可能丢失消息的场景有下面这些。

在第一部分中。步骤 1、2、3 都可能存在失败的情况。

由于用户 A 发消息是一个“请求”和“响应”的过程，如果用户 A 在把消息发送到 IM 服务器的过程中，由于网络不通等原因失败了；或者 IM 服务器接收到消息进行服务端存储时失败了；或者用户 A 等待 IM 服务器一定的超时时间，但 IM 服务器一直没有返回结果，那么这些情况用户 A 都会被提示发送失败。

接下来，他可以通过重试等方式来弥补，注意这里可能会导致发送重复消息的问题。

比如：客户端在超时时间内没有收到响应然后重试，但实际上，请求可能已经在服务端成功处理了，只是响应慢了，因此这种情况需要服务端有去重逻辑，一般发送端针对同一条重试消息有一个唯一的 ID，便于服务端去重使用。

在第二部分中。消息在 IM 服务器存储完后，响应用户 A 告知消息发送成功了，然后 IM 服务器把消息推送给用户 B 的在线设备。

在推送的准备阶段或者把消息写入到内核缓冲区后，如果服务端出现掉电，也会导致消息不能成功推送给用户 B。

这种情况实际上由于连接的 IM 服务器可能已经无法正常运转，需要通过后期的补救措施来解决丢消息的问题，后续会详细讲到，这里先暂且不讨论。

即使我们的消息成功通过 TCP 连接给到用户 B 的设备，但如果用户 B 的设备在接收后的处理过程出现问题，也会导致消息丢失。比如：用户 B 的设备在把消息写入本地 DB 时，出现异常导致没能成功入库，这种情况下，由于网络层面实际上已经成功投递了，但用户 B 却看不到消息。所以比较难处理。

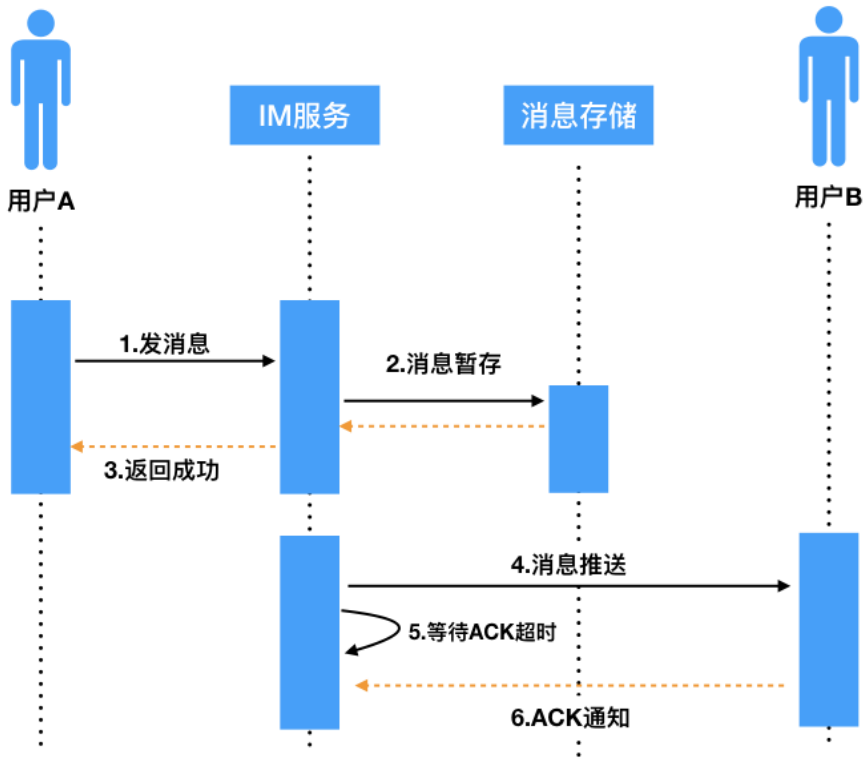
上面两种情况都可能导致消息丢失，那么怎么避免这些异常情况下丢消息的问题呢？
一般我们会用下面这些相应的解决方案：

1. 针对第一部分，我们通过客户端 A 的超时重发和 IM 服务器的去重机制，基本就可以解决问题；
2. 针对第二部分，业界一般参考 TCP 协议的 ACK 机制，实现一套业务层的 ACK 协议。

解决丢失的方案：业务层 ACK 机制

我们先解释一下 ACK，ACK 全称 Acknowledge，是确认的意思。在 TCP 协议中，默认提供了 ACK 机制，通过一个协议自带的标准的 ACK 数据包，来对通信方接收的数据进行确认，告知通信发送方已经确认成功接收了数据。

那么，业务层 ACK 机制也是类似，解决的是：IM 服务推送后如何确认消息是否成功送达接收方。具体实现如下图：



IM 服务器在推送消息时，携带一个标识 SID（安全标识符，类似 TCP 的 sequenceId），推送出消息后会将当前消息添加到“待 ACK 消息列表”，客户端 B 成功接收完消息后，会给 IM 服务器回一个业务层的 ACK 包，包中携带有本条接收消息的 SID，IM 服务器接收后，会从“待 ACK 消息列表”记录中删除此条消息，本次推送才算真正结束。

ACK 机制中的消息重传

如果消息推给用户 B 的过程中丢失了怎么办？比如：

B 网络实际已经不可达，但 IM 服务器还没有感知到；

用户 B 的设备还没从内核缓冲区取完数据就崩溃了；

消息在中间网络途中被某些中间设备丢掉了，TCP 层还一直重传不成功等。

以上的问题都会导致用户 B 接收不到消息。

解决这个问题的常用策略其实也是参考了 TCP 协议的重传机制。类似的，IM 服务器的“等待 ACK 队列”一般都会维护一个超时计时器，一定时间内如果没有收到用户 B 回的

ACK 包，会从“等待 ACK 队列”中重新取出那条消息进行重推。

消息重复推送的问题

刚才提到，对于推送的消息，如果在一定时间内没有收到 ACK 包，就会触发服务端的重传。收不到 ACK 的情况有两种，除了推送的消息真正丢失导致用户 B 不回 ACK 外，还可能是用户 B 回的 ACK 包本身丢了。

对于第二种情况，ACK 包丢失导致的服务端重传，可能会让接收方收到重复推送的消息。

针对这种情况，一般的解决方案是：服务端推送消息时携带一个 Sequence ID，Sequence ID 在本次连接会话中需要唯一，针对同一条重推的消息 Sequence ID 不变，接收方根据这个唯一的 Sequence ID 来进行业务层的去重，这样经过去重后，对于用户 B 来说，看到的还是接收到一条消息，不影响使用体验。

这样真的就不会丢消息了吗?

细心的你可能发现，通过“ACK+ 超时重传 + 去重”的组合机制，能解决大部分用户在线时消息推送丢失的问题，那是不是就能完全覆盖所有丢消息的场景呢?

设想一下，假设一台 IM 服务器在推送出消息后，由于硬件原因宕机了，这种情况下，如果这条消息真的丢了，由于负责的 IM 服务器宕机了无法触发重传，导致接收方 B 收不到这条消息。

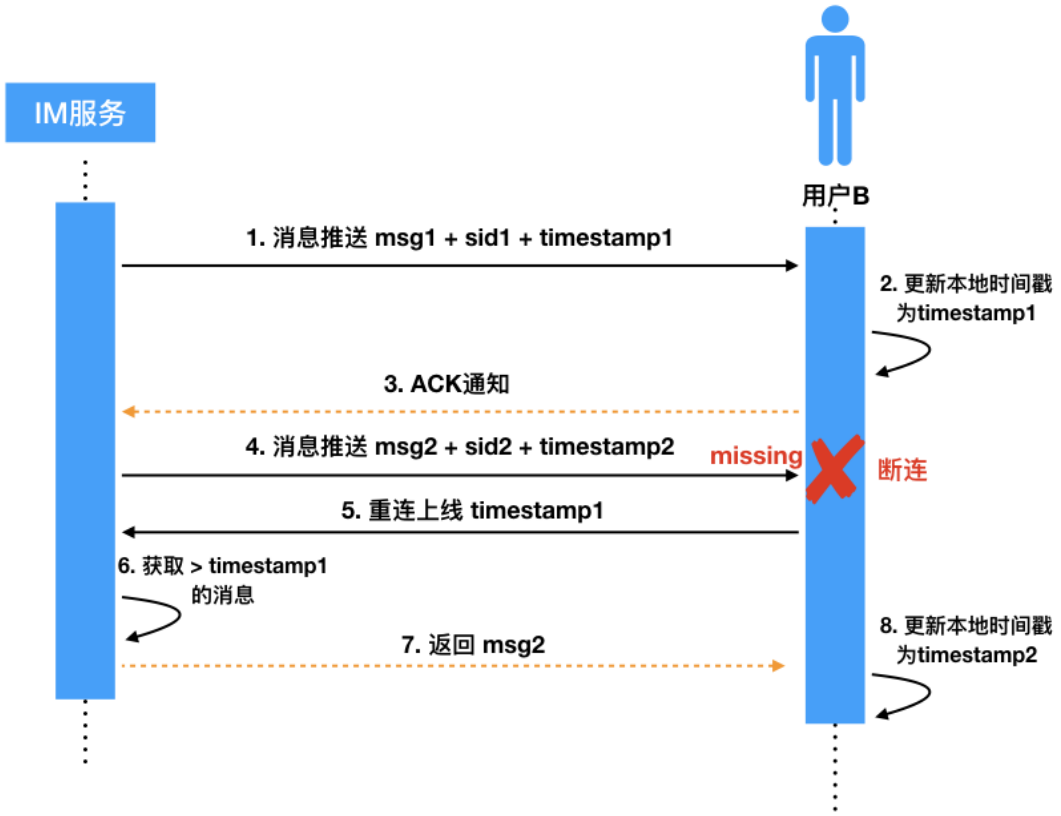
这就存在一个问题，当用户 B 再次重连上线后，可能并不知道之前有一条消息丢失的情况。对于这种重传失效的情况该如何处理?

补救措施：消息完整性检查

针对服务器宕机可能导致的重传失效的问题我们来分析一下，这里的问题在于：服务器机器宕机，重传这条路走不通了。

那如果在用户 B 在重新上线时，让服务端有能力进行完整性检查，发现用户 B“有消息丢失”的情况，就可以重新同步或者修复丢失的数据。

比较常见的消息完整性检查的实现机制有“时间戳比对”，具体的实现如下图：



下面我们来看一下“时间戳机制是如何对消息进行完整性检查的，我用这个例子来解释一下这个过程。

IM 服务器给接收方 B 推送 msg1，顺便带上一个最新的时间戳 timestamp1，接收方 B 收到 msg1 后，更新本地最新消息的时间戳为 timestamp1。

IM 服务器推送第二条消息 msg2，带上一个当前最新的时间戳 timestamp2，msg2 在推送过程中由于某种原因接收方 B 和 IM 服务器连接断开，导致 msg2 没有成功送达到接收方 B。

用户 B 重新连上线，携带本地最新的时间戳 timestamp1，IM 服务器将用户 B 暂存的消息中时间戳大于 timestamp1 的所有消息返回给用户 B，其中就包括之前没有成功的 msg2。

用户 B 收到 msg2 后，更新本地最新消息的时间戳为 timestamp2。

通过上面的时间戳机制，用户 B 可以成功地让丢失的 msg2 进行补偿发送。

需要说明的是，由于时间戳可能存在多机器时钟不同步的问题，所以可能存在一定的偏差，导致数据获取上不够精确。所以在实际的实现上，也可以使用全局的自增序列作为版本号来代替。

小结

保证消息的可靠投递是 IM 系统设计中至关重要的一个环节，“不丢消息”“消息不重复”对用户体验的影响较大，我们可以通过以下手段来确保消息下推的可靠性。

- 大部分场景和实际实现中，通过业务层的 ACK 确认和重传机制，能解决大部分推送过程中消息丢失的情况。
- 通过客户端的去重机制，屏蔽掉重传过程中可能导致消息重复的问题，从而不影响用户体验。
- 针对重传消息不可达的特殊场景，我们还可以通过“兜底”的完整性检查机制来及时发现消息丢失的情况并进行补推修复，消息完整性检查可以通过时间戳比对，或者全局自增序列等方式来实现。

最后，给你留一个思考题，有了 TCP 协议本身的 ACK 机制为什么还需要业务层的 ACK 机制？

你可以给我留言，我们一起讨论，感谢你的收听，我们下期再见。

上一篇　03 | 轮询与长连接：如何解决消息的实时到达问题?

下一篇　05 | 消息序号生成器：如何保证你的消息不会乱序?

精选留言 (34)

写留言

王棕生
2019-09-04

有了 TCP 协议本身的 ACK 机制为什么还需要业务层的ACK 机制?
答：这个问题从操作系统(linux/windows/android/ios)实现TCP协议的原理角度来说明更合适：
　1 操作系统在TCP发送端创建了一个TCP发送缓冲区，在接收端创建了一个TCP接收缓冲区；...
展开

作者回复: 👍，嗯，即使数据成功发送到接收方设备了，tcp层再把数据交给应用层时也可能出现异常情况，比如存储客户端的本地db失败，导致消息在业务层实际是没成功收到的。这种情况下，可以通过业务层的ack来提供保障，客户端只有都执行成功才会回ack给服务端。

1 25

小可
2019-09-04

两个ack的作用不同，tcp的ack表征网络层消息是否送达；业务层ack是真正的业务消息是否送达和是否正确处理，达到不丢消息，消息不重复的目的，即我们要保证的消息可靠性

作者回复: 👍

5

墙角儿的花
2019-09-04

1、回答老师的问题：TCP层的ACK只是TCP包分片的ACK，并不能代表整个应用层的消息得到应答。理论上操作系统的TCP栈肯定是知道整个TCP消息得到对方的ACK了，但是操作系统好像并没提供这种接口。发送成功的接口返回成功通常都表示为操作系统发送成功了，至于链路上有没有问题就不知道了。
2、向老师请教下其他问题，恳请解答。...
展开

作者回复: A. 接收方本地去重只需要针对本机已经接收到的存在的消息来做就可以了，服务端接收时实际上已经会做一次存储层的去重了，只会存在没有回ack的消息导致接收方重复接收的情况，这种两次之间一般时间间隔都比较短的。
B. 如果低序号的消息还没到，由于没有收到客户端的ack服务端会有超时重传机制会重传这条低序号的消息，另外即使这个时候用户关机不等那条消息了，再次上线时，采用版本号机制的话客户端也是可以知道消息不完整，可以触发服务端进行重推。

2 3

隰有荷
2019-09-04

您好，我在读到在消息完整性检查那里时有些疑惑，如果服务端将msg2发出之后，服务端和客户端断链，导致客户端无法接收消息，那么重新连接之后，是可以发送时间戳检测进行重传的。
但是，如果在服务端存储了发送方客户端发送的消息后，正准备将该消息推送给接收方客户端时发生宕机，那么当接收方客户端和服务端重新连接之后，服务端该如何知道自己...
展开

作者回复: 用户上线的时候携带本地最新一条消息的时间戳给服务端，服务端从离线缓存里取比这个时间戳大的消息发给客户端就行了呀

2

长江
2019-09-04

有了 TCP 协议本身的 ACK 机制为什么还需要业务层的 ACK 机制?
1.TCP属于传输层，而IM服务属于应用层，TCP的ACK只能保证传输层的可靠性，即A端到B端的可靠性，但是不能保证数据能够被应用层正确可靠处理，比如应用层里面的业务逻辑导致消息处理失败了，TCP层是不知道的。
2.TCP虽然是可靠性传输协议，但是如果传输过程中，假如数据报文还没被接收端接收完...
展开

作者回复: 1没问题哈，对于2的话如果出现接收端进程崩溃，一般这个时候接收端APP也是处于不可用状态了，这种情况实际上也没法通过这个ACK机制来避免丢消息。可以在用户再次上线时进行完整性检查的确认，如果有消息没有被正确接收，再由服务补推。

4 2

卫江
2019-09-04

Tcp的ack机制可以保证通过tcp传输的数据被对端内核接受并放入对应的socket接受缓存区里面，但是接下来进程读取缓存区以及进行逻辑处理可能会出现问题，所以需要应用层的ack机制保证数据包被进程读取并正确的处理。
展开

1

RuBy
2019-09-04

老师，请问消息落地的话传统的redis+mysql是否会有性能瓶颈？是否会考虑leveldb（rocksdb）这种持久化kv存储呢?
展开

作者回复: 看量级吧，我们自己的场景里mysql和hbase做为永久存储，pika作为离线消息的buffer存储 没有碰到瓶颈。

1

阳阳
2019-09-04

请问一下老师：如果im服务器发给用户的第一条消息，用户b没有给业务层ack，但是im服务器紧接着又收到了第二条消息往用户b发送，此时用户b返回了正常ack。那么第一条消息在超时的时候会从队列里拿出来重传，假设用户b收到且返回ack成功，这个时候第一条消息

展开

作者回复: 留言没完整哈，如果是重复接收的问题，需要客户端来进行去重的。

1

Better me

2019-09-04

老师您好，我想问下服务端IM先后推送两条消息msg0、msg1、msg2到客户端B，如果msg0、msg2先到达，此时客户端B应该不会更新到msg2的发送时间戳吧，而是等待msg1到达。如果此时陆续msg3、msg4消息到达，msg1还没到达，客户端是否也可像tcp一样，发送3个msg0消息到服务端IM，而不需要等待超时就让IM服务端立即重发msg1，这样可以降低延时，等到msg1到达客户端B后，可以直接将时间戳更新至最新到达消息的...

展开

作者回复: 首先这里要能让客户端知道有一条消息msg1还没到，所以单纯使用时间戳可能是不够的，时间戳只能代表时序，并不能判断出完整性，所以可能还需要配合连续的序号来实现感知（比如每次建连，服务端针对这条连接的所有推送从0开始自增，随消息一起下推）。至于你说的类似tcp的重传机制，这个理论上是可以这么实现的，就是复杂度上会高一些。

1

段先森

2019-09-04

想问一下老师，一般来说对于直播的业务场景，消息存储这个环节用什么MQ会比较好些

作者回复: 如果只是离线消息暂存的话，可以用pika或者redis，如果是消息的永久存储还是落db比较好。

1

小袁

2019-09-09

有2个问题

1 服务端发消息给客户端，没收到ack重试有最大次数吗？是像tcp那样一直重试直到收到a

ck吗?

如果客户端有bug导致处理消息一直失败，是否会导致服务端一直在重试，然后队列就满...

展开

作者回复: 1. 会有重试次数，毕竟即使收不到还有离线消息来补充。
重试多次仍然失败服务端可以主动断连来避免资源消耗。
2. 一般等着服务端重推就好了。

钢
2019-09-07

如果客户端一直连接不上，IM服务器会怎么处理，缓存一段时间的数据?

作者回复: 对，一般至少会缓存到离线消息的buffer中。

YidWang
2019-09-06

我认为时间戳做这件事 不正确，链路交互太长 失败率太高!

作者回复: 链路长不是问题呀，只要这个 时间相关的序号 可以一直跟着消息落db的话。

Chao
2019-09-06

tcp ack 。只保证分组的数据完整，不保证客户端传输的完整

作者回复: tcp的ack只能保证连接层的“不丢数据”和“不乱序”，但数据在接收时给到应用层的时候还可能出现丢失的情况。

东东
2019-09-05

在线群聊ack没有懂，老师能举个实例吗，比如在线群消息客户端插入失败然后怎么处理呢？谢谢

作者回复: 比如群里有用户A、B、C，这三个用户分别连接到IM服务器，A往群里发送一条消息，B和C的连接的网关机分别下推这一条消息，并且将这条消息分别加入到B和C的“待ACK列表”，如果B接收后处理成功然后回了ACK，B的网关机收到ACK就会从“待ACK列表”删除这条消息；如果C接收后客户端本地db写入失败，这时C的客户端就不会回ACK，C的网关机一段时间后没有接收到这条消息的ACK，就会触发超时重传，重新发送这条消息给C的客户端。

1

Geek_37e993
2019-09-05

设想一下，假设一台 IM 服务器在推送出消息后，由于硬件原因宕机了，这种情况下，如果这条消息真的丢了，由于负责的 IM 服务器宕机了无法触发重传，导致接收方 B 收不到这条消息。

这就存在一个问题，当用户 B 再次重连上线后，可能并不知道之前有一条消息丢失的情...
展开

作者回复: 这个和服务端没有收到ack进行从“待ack列表”取出未ack的消息进行重推是类似处理思路。

一路向北
2019-09-05

我有个想法，关于服务端消息暂存。当服务端接收到消息后，先存到redis中（因为redis缓存速度比DB快），一旦存入redis成功，就同时反馈用户A，和推送用户B。然后再将消息从redis中通过异步操作存到DB中。这样对提升服务器性能有真正的帮助吗?
展开

作者回复: 嗯，实际上确实是可以这么做的。发送时只写缓存然后响应，来提升发送性能。不过一般推送用户B会在落地完DB后。

1

东东
2019-09-05

比如群里面5个人，一条群消息，如果只被3个人ack了，那这条消息还可以在等待ack队列里面进行重推？
还有消息完整性检查是推还是拉，有什么区别？
如果拉的话，是上线就分页一次拉取每个群的最新100条的话，群多的话会不会影响性能？
假如一个群，每天上万条消息，采用读扩散还是写扩散呢？...
展开

作者回复: 1. ack不是和某一条消息绑定的，是和某一个人要接收的消息绑定的，是某一个连接维度的。
2. 可以只拉取联系人第一页中的群的第一页的消息。
3. 一般是读扩散。

FF
2019-09-05

几个问题。问题一，IM 服务器和接收端 B 之前的网络如果是硬件类的故障，长时间连不上，这种情况下 IM 服务器是重推到成功为止吗？ 如果受影响的 B 是大范围，像支付宝之前的光缆被挖断，那 IM 服务器的压力不是特别大吗？ 考虑重推 N 次失败后不再重推，告诉用户重发这种方案如何呢？
...
展开

作者回复: 1. 一般尝试重推几次，还是失败的话就放弃。因为等用户再次上线是还是能够通过离线消息获取到的。
2. 服务端只能用来防止同一个用户的同一条消息发送多次的去重，对于推送过程中由于没有收到ack导致的接收重复需要接收端来做本地的去重，接收方去重和不保存用户消息不冲突。

小胡子
2019-09-05

应用层增加这么复杂的ack机制时，感觉影响推送服务的性能，这时应该有哪些优化手段

作者回复: ack本身对服务端来说开销还好呀，大部分消息到达率是很高的，所以一般只是从本地内存的”待ack列表“删除接收到ack对应的那条记录就行。如果确实对这个很敏感，也可以把ack逻辑独立到另外的服务去处理。