08 | 智能心跳机制：解决网络的不确定性

2019-09-13 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 16:52 大小 11.58M

你好，我是袁武林。

在前面的章节里，我讲到了在即时消息场景中非常重要的两个特性：“可靠投递”和“实时性”。

为了让消息能更加实时、可靠、快速地触达到接收方，大部分 IM 系统会通过“长连接”的方式来建立收发双方的通信通道，这些基于 TCP 长连接的通信协议，在用户上线连接时，会在服务端维护好连接到服务器的用户设备和具体 TCP 连接的映射关系，通过这种方式服务端也能通过这个映射关系随时找到对应在线的用户的客户端，而且这个长连接一旦建立，就一直存在，除非网络被中断。

因为“长连接”方式相比“短连接轮询”，不仅能节约不必要的资源开销，最重要的是能够通过“服务端推送”，提供更加实时的消息下发。

同样，对于发送方来说，如果发送消息也能通过“长连接”通道把消息给到 IM 服务端，相对于短连接方式，也能省略 TCP 握手和 TLS 握手的几个 RTT 的时间开销，在用户体验和实时性上也会更好。

为什么需要心跳机制

“长连接”方式给我们带来了众多好处，那么要让消息通过“长连接”实现可靠投递，最重要的环节就在于如何维护好这个“长连接”。

由于这个“长连接”底层使用的 TCP 连接并不是一个真正存在的物理连接，实际上只是一个无感知的虚拟连接，中间链路的断开连接的两端不会感知到，因此维护好这个“长连接”一个关键的问题在于能够让这个“长连接”能够在中间链路出现问题时，让连接的两端能快速得到通知，然后通过“重连”来重新建立新的可用连接，从而让我们这个“长连接”一直保持“高可用”状态。

这个能够“快速”“不间断”识别连接可用性的机制，被称为“心跳机制”。“心跳机制”通过持续的往连接上发送“模拟数据”来试探连接的可用性，同时也让我们的连接在没有真正业务数据收发的时候，也持续有数据流通，而不会被中间的网络运营商以为连接已经没有在使用而把连接切断。

下面，我会从两个方面带你详细了解一下心跳机制在长连接维护中的必要性。

降低服务端连接维护的开销

首先，心跳机制可以让 IM 服务端能尽快感知到连接的变化，从而尽早清理服务端维护连接使用的资源。

对于大部分即时通讯场景，消息收发双方经常处于移动网络环境中，手机信号强弱变化及中间路由故障等，都可能导致“长连接”实际处于不可用状态。

比如：用户拿着手机进电梯了，手机网络信号忽然完全没了，长连接此时已经不可用，但 IM 服务端无法感知到这个“连接不可用”的情况；另外，假如我们上网的路由器忽然掉线

了，之前 App 和 IM 服务端建立的长连接，此时实际也处于不可用状态，但是客户端和 IM 服务器也都无法感知。

我在前面讲过，之所以能够实现消息的“服务端推送”，是因为我们针对每一台上线的设备，都会在 IM 服务端维护相应的“用户设备”和“网络连接”这么一个映射关系，除此之外，很多时候为了节省网络开销，还会在服务端临时缓存一些没必要每次请求都携带的客户端的信息（比如：app 版本号、操作系统、网络状态等），这样客户端一旦建好长连后，只需要首次携带这些信息，后续请求可以不用再携带，而是使用 IM 服务端缓存的这些信息。另外，在很多 IM 的实现上，还会在服务端维护一些“用户在线状态”和“所有在线设备”这些信息，便于业务使用。

如果 IM 服务端无法感知到这些连接的异常情况，会导致的一个问题是：IM 服务端可能维护了大量“无效的连接”，从而导致严重的连接句柄的资源浪费；同时也会缓存了大量实际上已经没有用了的“映射关系”“设备信息”“在线状态”等信息，也是对资源的浪费；另外，IM 服务端在往“无效长连接”推送消息，以及后续的重试推送都会降低服务的整体性能。

支持客户端断线重连

通过“心跳”快速识别连接的可用性，除了可以降低服务端的资源开销，也被用于来支撑客户端的断开重连机制。

对于客户端发出心跳包，如果在一定的超时时间内（考虑到网络传输具有一定的延迟性，这个超时时间至少要大于一个心跳的间隔），比如连续两次发送心跳包，都没有收到 IM 服务端的响应，那么客户端可以认为和服务端的长连接不可用，这时客户端可以断线重连。

导致服务端没有响应的原因可能是和服务端的网络在中间环节被断开，也可能是服务器负载过高无法响应心跳包，不管什么情况，这种场景下断线重连是很有必要的，它能够让客户端快速自动维护连接的可用性。

连接保活

维护一条“高可用”的长连接，还有一个重要的任务就是**尽量让建立的长连接存活时间更长。**

这里你可能会问：难道在用户网络和中间路由网络都正常的情况下，长连接还可能会被杀死？

答案是：确实会。

探究这个原因的话，我可能要从 IPv4 说起。由于 IPv4 的公网 IP 的资源有限性（约 43 亿个），为了节省公网 IP 的使用，通过移动运营商上网的手机实际上只是分配了一个运营商内网的 IP。

在访问 Internet 时，运营商网关通过一个“外网 IP 和端口”到“内网 IP 和端口”的双向映射表，来让实际使用内网 IP 的手机能和外网互通，这个网络地址的转换过程叫做 NAT（Network Address Translation）。

NAT 本身的实现机制并没有什么不妥，问题在于很多运营商为了节省资源和降低自身网关的压力，对于一段时间没有数据收发的连接，运营商会将它们从 NAT 映射表中清除掉，而且这个清除动作也不会被手机端和 IM 服务端感知到。

这样的话，如果没有了 NAT 映射关系，长连接上的消息收发都无法正常进行。而且多长时间会从 NAT 映射表清除，每个地方的运营商也是不尽相同，从几分钟到几小时都有。假设用户有几分钟没有收发消息，可能这个长连接就已经处于不可用状态了。

那么，如果我们的客户端能在没有消息收发的空闲时间给服务端发送一些信令，就能避免长连接被运营商 NAT 干掉了，这些“信令”一般就是通过心跳包来实现。

心跳检测的几种实现方式

介绍完了心跳机制的重要性，我们来学习一下如何去实现心跳检测。目前业界有三种常用的实现方法：TCP Keepalive、应用层心跳及智能心跳。下面我们分别来看一看。

TCP Keepalive

TCP 的 Keepalive 作为操作系统的 TCP/IP 协议栈实现的一部分，对于本机的 TCP 连接，会在连接空闲期按一定的频次，自动发送不携带数据的探测报文，来探测对方是否存活。操作系统默认是关闭这个特性的，需要由应用层来开启。

默认的三个配置项：心跳周期是 2 小时，失败后再重试 9 次，超时时间 75s。三个配置项均可以调整。

这样来看，TCP 的 Keepalive 作为系统层 TCP/IP 协议栈的已有实现，不需要其他开发工作量，用来作为连接存活与否的探测机制是非常方便的；上层应用只需要处理探测后的连接异常情况即可，而且心跳包不携带数据，带宽资源的浪费也是最少的。

由于易用性好、网络消耗小等优势，TCP Keepalive 在很多 IM 系统中被开启使用，之前抓包就发现，WhatsApps 使用空闲期 10 秒间隔的 TCP Keepalive 来进行存活探测。

虽然拥有众多优势，但 TCP Keepalive 本身还是存在一些缺陷的，比如心跳间隔灵活性较差，一台服务器某一时间只能调整为固定间隔的心跳；另外 TCP Keepalive 虽然能够用于连接层存活的探测，但并不代表真正的应用层处于可用状态。

我举一个例子，比如 IM 系统出现代码死锁、阻塞的情况下，实际上已经无法处理业务请求了，但此时连接层 TCP Keepalive 的探针不需要应用层参与，仍然能够在内核层正常响应。这种情况就会导致探测的误判，让已失去业务处理能力的机器不能被及时发现。

应用层心跳

为了解决 TCP Keepalive 存在的一些不足的问题，很多 IM 服务使用应用层心跳来提升探测的灵活性和准确性。**应用层心跳实际上就是客户端每隔一定时间间隔，向 IM 服务端发送一个业务层的数据包告知自身存活。**

如果 IM 服务端在一定时间内没有收到心跳包，就认定客户端由于某种原因连接不可达了，此时就会从 IM 服务端把这个连接断开，同时清除相应分配的其他资源。

应用层心跳和 TCP Keepalive 心跳相比，由于不属于 TCP/IP 协议栈的实现，因此会有一些额外的数据传输开销，但是大部分应用层心跳的设计上心跳包都尽量精简，一般就几个字节，比如有些应用层心跳包只是一个空包用于保活，有的心跳包只是携带了心跳间隔，用于客户端调整下一次的心跳，所以额外的数据开销都非常小。

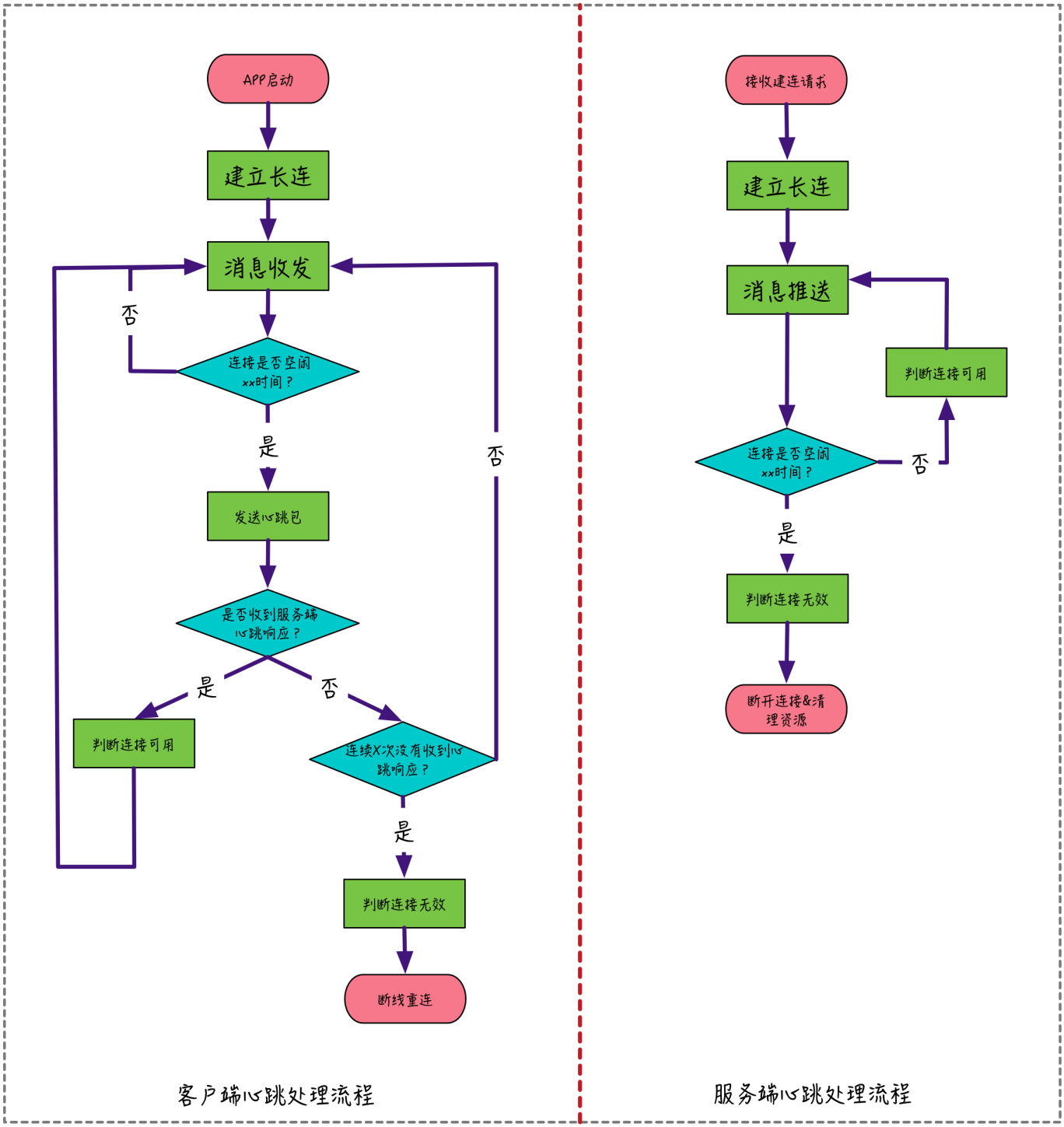
应用层心跳相比 TCP Keepalive，由于需要在应用层进行发送和接收的处理，因此更能反映应用的可用性，而不是仅仅代表网络可用。

而且应用层心跳可以根据实际网络的情况，来灵活设置心跳间隔，对于国内运营商 NAT 超时混乱的实际情况下，灵活可设置的心跳间隔在节省网络流量和保活层面优势更明显。

目前大部分 IM 都采用了应用层心跳方案来解决连接保活和可用性探测的问题。比如之前抓包中发现 WhatApps 的应用层心跳间隔有 30 秒和 1 分钟，微信的应用层心跳间隔大部分情况是 4 分半钟，目前微博长连接采用的是 2 分钟的心跳间隔。

每种 IM 客户端发送心跳策略也都不一样，最简单的就是按照固定频率发送心跳包，不管连接是否处于空闲状态。之前抓手机 QQ 的包，就发现 App 大概按照 45s 的频率固定发心跳；还有稍微复杂的策略是客户端在发送数据空闲后才发送心跳包，这种相比较对流量节省更好，但实现上略微复杂一些。

下面是一个典型的应用层心跳的客户端和服务端的处理流程图，从图中可以看出客户端和服务端，各自通过心跳机制来实现“断线重连”和“资源清理”。



需要注意的是：对于客户端来说，判断连接是否空闲的时间是既定的心跳间隔时间，而对于服务端来说，考虑到网络数据传输有一定的延迟，因此判断连接是否空闲的超时时间需要大于心跳间隔时间，这样能避免由于网络传输延迟导致连接可用性的误判。

智能心跳

在国内移动网络场景下，各个地方运营商在不同的网络类型下 NAT 超时的时间差异性很大。采用固定频率的应用层心跳在实现上虽然相对较为简单，但为了避免 NAT 超时，只能将心跳间隔设置为小于所有网络环境下 NAT 超时的最短时间，虽然也能解决问题，但对于设备 CPU、电量、网络流量的资源无法做到最大程度的节约。

为了优化这个现象，很多即时通讯场景会采用“智能心跳”的方案，来平衡“NAT 超时”和“设备资源节约”。所谓智能心跳，就是让心跳间隔能够根据网络环境来自动调整，通过不断自动调整心跳间隔的方式，逐步逼近 NAT 超时临界点，在保证 NAT 不超时的情况下尽量节约设备资源。据说微信就采用了智能心跳方案来优化心跳间隔。

不过从个人角度看，随着目前移动资费的大幅降低，手机端硬件设备条件也越来越好，智能心跳对于设备资源的节约效果有限。而且智能心跳方案在确认 NAT 超时临界点的过程中，需要不断尝试，可能也会从一定程度上降低“超时确认阶段”连接的可用性，因此，我建议你可以根据自身业务场景的需要，来权衡必要性。

小结

简单回顾一下今天的内容：为了保证消息下发的实时性，很多即时通讯场景使用“长连接”来降低每次建立连接消耗的时间，同时避免了“短连接轮询”带来的不必要的资源浪费。

但是，由于移动网络环境错综复杂，网络状态变化、中间链路断开、运营商 NAT 超时都可能导致这个“长连接”处于不可用状态，而且收发双发无法感知到。

通过客户端和 IM 服务端建立的“心跳机制”可以快速自动识别连接是否可用，同时避免运营商 NAT 超时被断开的情况。“心跳机制”解决了以下三方面的问题：

降低服务端连接维护无效连接的开销。

支持客户端快速识别无效连接，自动断线重连。

连接保活，避免被运营商 NAT 超时断开。

心跳探测的实现业界大部分综合采用以下两种方式：

TCP Keepalive。操作系统 TCP/IP 协议栈自带，无需二次开发，使用简单，不携带数据网络流量消耗少。但存在灵活性不够和无法判断应用层是否可用的缺陷。

应用层心跳。应用自己实现心跳机制，需要一定的代码开发量，网络流量消耗稍微多一点，但心跳间隔的灵活性好，配合智能心跳机制，可以做到”保证 NAT 不超时的情况下最大化节约设备资源消耗 “，同时也能更精确反馈应用层的真实可用性。

最后给大家留一道思考题：

心跳机制中可以结合 TCP 的 keepalive 和应用层心跳来一起使用吗？

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。

上一篇　07 | 分布式锁和原子性：你看到的未读消息提醒是真的吗？

下一篇　09 | 分布式一致性：让你的消息支持多终端漫游？

精选留言 (10)

写留言

钢
2019-09-13

tcp keepalive解决网络可用性，应用层keepalive解决网络和服务可用性，应用层无法区别网络还是服务问题，加上tcp心跳包可以进一步区分，有助于客户端做动态调整

7

newzai
2019-09-13

智能心跳，采用什么算法逼近或者选择心跳周期？如何感知当前的心跳周期是否太小？因为一旦太大就会导致连接断开

展开

2

Geek_e986e3
2019-09-15

智能心跳的算法是咋样的

展开

1

王棕生
2019-09-14

解答问题：如果从实现功能角度看，传输层和应用层的心跳机制没有结合的必要，因为传输层的心跳探测连接可用性，应用层的心跳机制也可以完成探测; 但从debug角度看，应用层的心跳探测机制无法定位是网络的问题还是系统的问题，此时由传输层辅助就非常好，但实现会相对复杂!

展开

1

云师兄
2019-09-13

服务端在进行推送时候，假设判断用户连接有效，即用户在线，此时推送是同时推与客户端应用保持的长连接和apns等多条通道吗？然后在客户端去重？还是每次只会选择一条通道?

1

A:春哥大魔王
2019-09-14

老师 客户端或服务端心跳实现是在客户端维持一个轮训服务吗？这样会不会有浪费客户端资源的问题，有没有什么好方法解决轮训定时器这种资源浪费问题

展开

一步
2019-09-14

我看上图 心跳处理流程，只有客户端当判断连接无效的时候才会进行 连接重连，重连的这个动作服务端不会进行吗？ 如果这样的话服务端还要发送心跳消息进行心跳检测做什么呢？服务端是不是可以这样处理，当一段时间没有收到客户端的心跳包的时候就判断与该客户端的心跳连接断开了，直接释放对应的资源不就可以了吗?

展开

lorancechen

2019-09-13

文章中指出的心跳监测的时间间隔是发送

展开

卫江

2019-09-13

问题：心跳机制是不是可以结合tcp的keepalive和应用层的心跳协议？当然可以，不过只使用应用层的心跳协议就可以了，因为一旦应用层的心跳出现问题意味着连接出问题或服务处理不过来，不论什么问题，它更能反应应用真实的整体情况，比如负载均衡与后端服务器之间的应用心跳就能更客观的反应应用情况而便于进一步处理，比如摘除这个后端服务器并报警，而如果只是使用tcp的keepalive的反应的问题就比较片面，只是针对于网...

展开

钢

2019-09-13

老师，有没有应用层心跳包的设计规范及要求的文章推荐的