云风的 BLOG

思绪来得快去得也快，偶尔会在这里停留

« Skynet 的一些改进和进展 | 返回首页 | 开发笔记(25)：改进的 RPC »

记录一个并发引起的 bug

今天发现 Skynet 消息处理的一个 bug ，是由多线程并发引起的。又一次觉得完全把多线程程序写对是件很不容易的事。我这方面经验还是不太够，特记录一下，备日后回顾。

Skynet 的消息分发是这样做的：

所有的服务对象叫做 ctx ，是一个 C 结构。每个 ctx 拥有一个唯一的 handle 是一个整数。

每个 ctx 有一个私有的消息队列 mq ，当一个本地消息产生时，消息内记录的是接收者的 handle ，skynet 利用 handle 查到 ctx ，并把消息压入 ctx 的 mq 。

ctx 可以被 skynet 清除。为了可以安全的清除，这里对 ctx 做了线程安全的引用计数。每次从 handle 获取对应的 ctx 时，都会对其计数加一，保证不会在操作 ctx 时，没有人释放 ctx 对象。

skynet 维护了一个全局队列，globalmq ，里面保存了若干 ctx 的 mq 。

这里为了效率起见（因为有大量的 ctx 大多数时间是没有消息要处理的），mq 为空时，尽量不放在 globalmq 里，防止 cpu 空转。

Skynet 开启了若干工作线程，不断的从 globalmq 里取出二级 mq 。我们需要保证，一个 ctx 的消息处理不被并发。所以，当一个工作线程从 globalmq 取出一个 mq ，在处理完成前，不会将它压回 globalmq 。

处理过程就是从 mq 中弹出一个消息，调用 ctx 的回调函数，然后再将 mq 压回 globalmq 。这里不把 mq 中所有消息处理完，是为了公平，不让一个 ctx 占用所有的 cpu 时间。当发现 mq 为空时，则放弃压回操作，节约 cpu 时间。

所以，产生消息的时刻，就需要执行一个逻辑：如果对应的 mq 不在 globalmq 中，把它置入 globalmq 。

需要考虑的另一个问题是 ctx 的初始化过程：

ctx 的初始化流程是可以发送消息出去的（同时也可以接收到消息），但在初始化流程完成前，接收到的消息都必须缓存在 mq 中，不能处理。我用了个小技巧解决这个问题。就是在初始化流程开始前，假装 mq 在 globalmq 中（这是由 mq 中一个标记位决定的）。这样，向它发送消息，并不会把它的 mq 压入 globalmq ，自然也不会被工作线程取到。等初始化流程结束，在强制把 mq 压入 globalmq （无论是否为空）。即使初始化失败也要进行这个操作。

问题的焦点在于：删除 ctx 不能立刻删除 mq ，这是因为 mq 可能还被 globalmq 引用。而 mq 中并没有记录 ctx 指针（保存 ctx 指针在多线程环境是很容易出问题的，因为你无非保证指针有效），而保存的是 ctx 的 handle 。

我之前的错误在于，我以为只要把 mq 的删除指责扔给 globalmq 就可以了。当 ctx 销毁的那一刻，检查 mq 是否在 globalmq 中，如果不在，就重压入 globalmq 。等工作线程从 globalmq 中取出 mq ，从其中的 handle 找不到配对的 ctx 后，再将 mq 销毁掉。

问题就在这里。handle 和 ctx 的绑定关系是在 ctx 模块外部操作的（不然也做不到 ctx 的正确销毁），无法确保从 handle 确认对应的 ctx 无效的同时，ctx 真的已经被销毁了。所以，当工作线程判定 mq 可以销毁时（对应的 handle 无效），ctx 可能还活着（另一个工作线程还持有其引用），持有这个 ctx 的工作线程可能正在它生命的最后一刻，向其发送消息。结果 mq 已经销毁了。

Skynet 这次在编写过程中，经历过一次大的改变：最早我是采用的一级消息队列，而不是现在的两级。但是一级队列很难同时保证消息的时序性和 ctx 消息处理模块不可被并行运行。在设计修改的过程中，我可能做出了许多不优雅的实现。上面的问题或许可以经过一次梳理，更简单的解决。

而我现在是这样做的：

当 ctx 销毁前，由它向其 mq 设入一个清理标记。然后在 globalmq 取出 mq ，发现已经找不到 handle 对应的 ctx 时，先判断是否有清理标记。如果没有，再将 mq 重放进 globalmq ，直到清理标记有效，在销毁 mq 。

云风 提交于 August 17, 2012 06:12 PM | 固定链接

COMMENTS

老大您好，正在学习您的代码。感觉ctx/mq的组合像是一个虚拟机，mq是指令，ctx是处理器。不知我的理解对不对，不过我觉得对象关系最好还是以mq为主。比如ctx->forward应该保存对方的mq指针，而不是对方的ctx的句柄。毕竟消息是发到对方的mq里面去的，而不是ctx。。另外mq里面可以加上引用计数，保存ctx指针。ctx可能倒不是很需要引用计数。现在好像是以ctx为主的设计，感觉有点别扭。

Posted by: Puqing | (11) February 19, 2014 06:02 PM

@David Xu
为什么会有锁反的问题?
工作线程从globalmq 中取出的mq和即将放入globalmq中的mq 不是同一个mq啊

Posted by: cpluser | (10) June 21, 2013 02:43 PM

写的很好学到了呵呵，我会经常关注你的呵呵！

Posted by: 不锈钢合页 | (9) September 14, 2012 04:08 PM

如果使用函數式編程會否就解決了？但始終要改變思路是很困難的事

Posted by: word-pdf | (8) August 25, 2012 10:54 AM

来看看， 信誓旦旦的认为正确了， 确实不正确的

Posted by: ctx | (7) August 22, 2012 01:45 PM

关于引用计数和释放我是在对象里加了个变量(m_bDestoryed)，如果需要释放对象就把它设置为TRUE，并不真正释放，在相应使用对象的位置检查。另外引用计数如果用C++就可以用个工具类自动增加释放。一点拙见，见笑

Posted by: netboy | (6) August 20, 2012 04:57 PM

@David Xu

我没有细想， 是因为实现上是先有的 ctx ，后来重构加的 mq 。

原来是一级队列不是两级。

mq 是后加的东西，就没让 mq 去管理 ctx 。

可能需要以后全盘考虑一下，才会发现不同的算法。

Posted by: Cloud | (5) August 17, 2012 10:21 PM

忘记说了，对mq引用计数，就可以解决锁反的问题。

Posted by: David Xu | (4) August 17, 2012 09:53 PM

如果globalmq需要锁，而mq也需要锁,就建立了一个层次。有锁反的问题：
当ctx向mq放入一个消息时，需要把mq放到globalmq，这个时候需要global mq lock, 如果你已经获得了mq lock,那么现在的次序就是 mq lock, 然后是global mq lock.
但是后台工作线程是需要先获取global mq lock，然后拾起一个mq，再获得这个mq lock，释放global mq lock.

看来ctx需要一个中间状态来释放mq lock，然后再获得global mq lock，再把mq放入global mq.

如果一个mq对应一个ctx,到是不一定需要handle,由handle映射成ctx,可能需要锁。不如直接把ctx指针放在mq中。如果ctx想与mq脱离，可以把ctx设置成NULL。 至于如何销毁mq,那就是最后看到mq引用计数的为0的人，销毁mq.

Posted by: David Xu | (3) August 17, 2012 09:45 PM

@Harold Chan 我猜用函数式编程就会有新的问题；而且实现相似的功能不一定简单

Posted by: zhanxw | (2) August 17, 2012 09:09 PM

如果使用函數式編程會否就解決了？但始終要改變思路是很困難的事

Posted by: Harold Chan | (1) August 17, 2012 07:04 PM

POST A COMMENT

非这个主题相关的留言请到：留言本

名字：

Email 地址：

为了验证您是人类，请将六加一的结果（阿拉伯数字七）填写在下面：

URL:

记住我的信息？

留言：
（不欢迎在留言中粘贴程序代码）

提交