云风的 BLOG
思绪来得快去得也快，偶尔会在这里停留

« sproto 的实现与评测 | 返回首页 | Unity3D asset bundle 格式简析 »

skynet 中如何实现邮件达到通知服务

skynet 中可以独立的业务都是以独立服务形式存在的。昨天和同事讨论如何实现一个邮件通知服务。

目前大概是这样的：有一个独立的邮件中心服务，它可以处理三条协议：

1. 向一个 mailbox 投递一封邮件。
2. 查询一个 mailbox 里有多少封邮件。
3. 收取 mailbox 里指定的一封邮件。

用户读了多少邮件没有放在邮件中心，而是记在玩家数据里的。

用户的界面上需要显示是否有几封未读邮件，如果有新邮件达到，这个数字会自动变更。你可以想像成 iOS 上的那种带数字的小红点。

当然，在 skynet 的设计惯例中，每个用户在服务器上有一个 agent 代理，所以我们不单独考虑和客户端数据交互的问题，而只用考虑 agent 如何和邮件中心的交互。

现在的做法是，在用户上线的时候，就去邮件中心查一次，比较邮件数量后知道是否有新邮件，然后推送给玩家。

在玩家特定的操作后，比如进出副本等，都会重新查询一次。如果玩家在一个场景停留太久，客户端也会定期发起查询请求。

如果邮件必须在新邮件达到时，立刻通知给玩家怎么办呢？那么系统中另外有个用户中心的服务。邮件服务可以把消息推送到那里；用户中心发现玩家不在线，就扔掉消息；如果在线就做消息推送。

我觉得这个方案有那么一点点不好，所以提出了我的想法。

首先我不希望邮件中心服务只处理请求，而不要对外发送消息。因为这样，就必须让邮件业务了解更多的外部知识。

其次，定期查询显得很愚笨，也多了很多无谓的查询，很容易造成处理能力过载。因为外界无法确定邮件服务的处理能力（可能涉及外部数据库的查询），查询频率高于处理频率必定造成过载，而过载很容易雪崩，尤其是发起查询是其它独立系统决定的。

我认为在 skynet 框架里，更合适的做法是，当玩家上线时，agent 向邮件中心发送一条查询，附带自己已读邮件数量。

如果没有新邮件到达，邮件中心就不回复这条查询请求，而不是回复一条没有新邮件。而 agent 不收到上一条查询的回应就不要提出下一条查询。

简单的做法是使用 skynet.fork 一个线程来 while ture 查询更新邮件数量。而玩家请求邮件数量时，只需要查询本地的变量即可，不要去邮件中心查询。

这样的好处是，一个查询者同时永远都只有一个查询请求。而如果查询对象变更频繁，也不会推送变更消息多次。对过载的防范要好的多。

但实现的复杂点在于，邮件中心需要在收到请求不能立刻回复时，要挂起回复操作，等新邮件达到再回复。

好在 skynet 做这个并不复杂，记录下请求的 source 和 session ，之后发送消息即可。我觉得这种模式很普遍，所以新增加了一个方便的 api skynet.response 来简化处理。和 skynet.ret 立刻回应消息不同，skynet.response 返回的是一个 closure 。需要回应消息的时候，调用它即可；而不需要在同一个 coroutine 里调用 skynet.ret 。

skynet.response 返回的函数，第一个参数是 true 或 false ，后面是回应的参数。当第一个参数是 false 时，会反馈给调用方一个异常；true 则是正常的回应。

这类方案之所以适合于 skynet 框架，是因为：创建业务线程，以及挂起请求，推迟回应这些，对于 skynet 都是非常廉价的操作。而 skynet 使用的粒度较小的 lua 沙盒可以高效的管理它们。

不过还是还有一个小问题。如果邮件中心是一个需要长期运行的服务，那么如果 agent 频繁上下线，发起 query 请求，而却一直没有新邮件达到的话，就会挂起很多空请求。当然，每个请求仅仅是几十字节的而外开销而已，通常不足为虑。如果你真的在乎，可以做一个定时器，每几个小时清理一次（没有新邮件也回应）即可。

或者 response 还有一个方法，第一个参数可以传入 "TEST" ，查询要回应的对象是否还存在，而不是真的把消息发出去。

其它的事情，skynet 框架已经做得很完备了。目前的机制是：

当 A call B 时，如果 B 在回应前就退出了，A 会收到一条异常，并正确的传播到 A 里的 call 调用处；

当 A call B ，而 B 在回应前，A 自己退出了，B 也会收到一条异常，提示 A 已经不在了。但不会影响 B 的执行流程，只是让框架回收一些必要的相关资源。

这种模式，可以用在很多场合。比如你可以用它来监控好友名单的上下线消息（好友在线状态）；你可以用来监控聊天频道的新消息（而不需要由聊天频道推送消息给你）；你可以用来监控你在拍卖行寄卖的东西有没有售出或是流拍，等等。

当一个模块是独立实现的时候，仅给出它可以给出的请求接口，而不定义这个模块可以向外推送的消息类型，是非常利于模块化的。

比如，在 skynet dev 分支上（打算在 0.6.0 版提供）有一个新特性叫 sharedata 。

它可以提供在同一个节点中，不同的服务共享一个结构数据。数据提供方可以发布这个数据的新版本，并通知给所有数据持有方更新版本。由于 sharedata 是基于内存共享的，数据提供方只是简单的把老版本数据的一个标志翻转为脏（不可逆转）。

这里我就使用了这个模式。因为 sharedata 是以库形式封装的，而数据的读方不必专门处理数据更新的消息。

注：在 erlang 中，每个服务都有一个 mailbox ，可以主动去过滤出需要的消息。skynet 没有这个机制，虽然它有另一个类似机制就是 message type ，但并不适合在这种场合用。因为多个库维护有限的 message type （上限 256 个）实在是太麻烦了。

我们也不想在任何读到数据脏标记时都去发起一个新数据获取的请求（这里会有比较复杂的异步调用的问题）；且在数据变更频繁时，我们永远只关心最后一版的数据。那么用这个模式就非常合适了。

代码见：lualib 下 sharedata.lua 中的 monitor 函数 ；服务提供方见 service 下 sharedatad.lua 中的 update 函数 。

云风 提交于 July 30, 2014 06:12 PM | 固定链接

COMMENTS

第一感对“如果没有新邮件到达，邮件中心就不回复这条查询请求，而不是回复一条没有新邮件。而 agent 不收到上一条查询的回应就不要提出下一条查询。”的做法比较不支持。感觉有点为了流量的节省而牺牲清晰度。

Posted by: zelor | (4) August 26, 2014 04:55 PM

刚好用skynet写完邮件系统才看到云大这篇文章，试下看看情况怎样！

Posted by: json | (3) August 5, 2014 11:46 AM

感觉机制上很像long pull，但是超时时间设得很长（数小时），同时用的服务端的agent来保持会话。
个人感觉稍有点过度设计，又不是消息量特别大的IM系统，没必要做这么复杂。如果怕数据库压力，做个缓存，或者做个counter计数器，压力就会小很多。

Posted by: natars | (2) July 31, 2014 01:20 AM

这，其实应该就是 Long poll 吧～

Posted by: zkc | (1) July 31, 2014 12:06 AM

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