云风的 BLOG
思绪来得快去得也快，偶尔会在这里停留

« lua-conf 让配置信息在不同的 lua 虚拟机间共享 | 返回首页 | skynet 的 snax 框架及热更新方案 »

对 skynet 的 gate 服务的重构

由于历史原因，skynet 中的 gate 服务最早是用 C 写的独立服务。后来 skynet 将 socket 的管理模块加入核心后又经历过一次重构，用后来增加的 socket api 重新编写了一遍。

目前，skynet 的各个基础设施逐步完善，并确定了以 lua 开发为主的基调，所以是时候用 lua 重写这个服务了。

如果是少量的连接且不关心性能的话，直接用 skynet 的 lua socket 库即可。这里有一个例子 。

gate 定位于高效管理大量的外部 tcp 长连接。它不是 skynet 的核心组件，但对于网络游戏业务，必不可少。

skynet 的内核已经集成了 epoll/kqueue 可以高效处理 socket 事件。但是离处理长连接还差一步，那就是对数据流的分包。

skynet 目前的 socket api ，采用回调的方式接收 socket 数据流。在一条 tcp 连接上，无论每次收到多少字节，都会使用 PTYPE_SOCKET 通道转发给绑定这条 tcp 连接的 skynet 服务。它是不关心数据流是如何组织的。

通常，我们会用 长度+数据内容 的形式对 TCP 数据流进行切分。这是在网络游戏中最常见的协议设计方案。

当然，也可以按 http 或其它流行网络协议(pop3 imap 等)那样，以回车换行符以及文本数字的方式来分包，但除了增加切包算法的复杂度外，没有太多好处。

目前 skynet 的 gate 服务约定的协议是，2 字节(大头编码）表示一个 64K 字节内的数据包，然后接下来就是这个长度的字节数。我曾经考虑过使用 4 字节或 google proto buffer 用的 varint ，但最后都放弃了。

考虑到实现的便捷，通常收到长度后，会在内存考虑指定长度的 buffer 等待后续的数据输入。这样，如果有大量攻击者发送超长包头，就会让服务器内存瞬间消进。所以，这种协议只要实现的不小心，很容易变成攻击弱点。

注：skynet 最早期的 gate 实现反而没有这个问题。因为它使用了单一的 ringbuffer ，只发送包头却不发送数据的连接会在 ringbuffer 回绕的时候被踢掉。

游戏服务器如果只使用一条 TCP 长连接的情况下，单个数据包过大（> 64K），也是不合适的。大包会阻塞应用逻辑（收取和发送它们都需要很长的时间），如果在应用层有心跳控制的话，也很容易造成心跳超时。所以一般在应用层对大数据包再做上层协议的切割处理。在本文的最后，会对此做一些讨论。

---

gate 的职责应该是保持大量 TCP 长连接，按协议切分。对于不完整的数据包，按连接分别置入独立的缓冲区中。对于每个完整的包，转发给需要的服务。

这里的工作分两部分，分包和转发。

分包以及对不完整的包做缓存是个细活，交个 C 代码去处理比较合适。但转发控制这部分业务比较复杂，lua 做更好。这就是这次重构的指导思想。

这次我把分包的工作放在了一个叫做 netpack 的 lua 扩展库里。参考：lua-netpack.c

然后 gate 的调度逻辑放在了 lua 版的 gate.lua 中。相较于上一版完全用 C 实现，会损失一点性能，但扩展性和可维护性都能提高很多。

最初在设计 gate 的时候，希望可以把多个连接上的数据转发给一个服务处理。比如你想用一个认证服务处理所有连接的最初登陆流程。又不想对网络数据包再打包（加上连接号），因为这样会造成额外的开销。

为了让接受数据的服务区分不同的数据源，我制作了一个代理服务 service_client 用来发送数据。并且在转发的时候，伪造这个代理服务作为数据源（真实的数据来源于 TCP 连接）。这样，处理数据的服务只需要按来源回应数据包就可以让网络数据包正确的返回。

而且，这种做法可以将 TCP 连接上的数据包通过 skynet.filter 包装成 skynet 内部消息格式。即，收到网络数据包(PTYPE_CLIENT) 时，数据包内其实包含有必要的 session 号，先把 session 和其他数据分离，通过 skynet.filter 分别传给下游。这样就把网络连接从业务层中屏蔽掉了。

这也是为什么 gate 的转发协议需要提供两个服务地址的原因。

这次新写的 gate 继承了这个用法。但过往的实践中，这个用法略显复杂。如果业务简单，其实用不着实现这么多配套服务。直接把 socket fd 交给业务处理的服务，它直接向 socket 发包即可。

重新写的 examples 里的 agent 就按这个思路实现。

examples 展示了简单的客户端服务器通讯协议的封装方法：消息主体使用 json 。在主体前面加上文本的 数字 session 加一个符号 + 或 - 。+ 表示这是一条请求消息，- 表示这条消息是对前面对应 session 号的消息的回应。

由于直接调用 socket api 发送数据包，所以 agent 不再需要和 service_client 配套使用。

原来 C 编写的 client demo 已经删除，换成了 lua 版的 client demo 。网络层使用了一个简单的 socket 库。如果用于实际项目，还需要完善 socket 库（客户端也不一定用 lua 实现）。

这次 examples 只是简单的重新理了一下代码。它还远远不是一个有复杂业务的 demo 。我一直在考虑到底提供一个怎样的 demo 可以完整的展示 skynet 的特点。目前还没有想好。

对于我们已经上线和即将上线的项目，结构比这个 example 要复杂的多。

首先我们使用的是 google proto buffer 协议，作为消息主体。但外围做了一些封装。消息由消息类型、session 消息主体构成。消息类型用自定义的小语言描述，用于消息分发。session 对应 skynet 内部的 session 号。根据消息类型，可以知道消息主体如何编码。

其次，每个连接建立后，不会立刻创建 agent 和它对接，因为这可能使登陆流程（尤其是未完成的登陆流程）给系统造成过大的负载。登陆过程的交互统一转给认证服务处理。认证服务是无状态的，所以可以在系统内启动多份以提高处理效率。认证结束后，才真正创建 agent 和连接对接。

创建 agent 的过程可能比较慢 ，所以我们会在服务启动的时候预先创建好数千个 agent 待用。有一个 agent pool 服务管理这些备用 agent 。一旦系统有空闲，就会不断补充备用。

我们在 TCP 连接做了进一步的加密处理，目前是对 gate 做了一些改造完成的。由于 gate 并非 skynet 核心模块，所以可以复制一份出来定制需要的功能。将来还希望加上断线重连的特性。

下一步，我希望给 skynet 的 socket 层加上低优先级数据包的队列。就是说，从现在的单一队列改成两个。如果你需要启用第二队列，那么这将是一个低优先级的发送队列。socket 发送规则如下：

1. 如果 socket 可写，且两个队列都为空，立即发送。
2. 如果上一次有一个数据包没有发送完，无论它在哪个队列里，都确保先将其发送完。
3. 如果高优先级队列有数据包，先保证发送高优先级队列上的数据。
4. 如果高优先级队列为空，且低优先级队列不为空，发送一个低优先级队列上的数据包。

这可以用来解决前面提到的大数据包的问题。

在应用层，我们可以把大数据包分割成不大于 4K 的小数据包。给大数据包添加一个唯一的编号。这些分割后的小数据包进入低优先级队列，那么它们就会被切碎传输给对端。一个大数据包可能被切成数百份，其它的数据会穿插再其间。尤其不会影响心跳控制包的传输。

例如，客户端向服务器请求拍卖行目录，或是全球排行榜这种数据量很大的数据时，走这个大数据包通道，就不会影响正常的交互流程了。

云风 提交于 April 15, 2014 11:32 AM | 固定链接

COMMENTS

@聪聪
都可以的，"Globle" ".NAME" ":HANDLE"

Posted by: thinka | (7) April 21, 2014 01:11 PM

@Cloud

相同的模块虽然能启动多份，貌似现在的接口skynet.call(addr, typename, ...)中，addr只能是模块的名字，无法使用address id。求实例

Posted by: 聪聪 | (6) April 16, 2014 07:18 PM

没有特别限制相同模块启动多少份.

活动的模块(service) 是用 address id 区分的.

Posted by: Cloud | (5) April 16, 2014 02:13 PM

这个架构能实现这个功能么：
一个模块一个server，我们这边需要启动N个这个样相同功能的模块，但是不支持，因为模块是易名字来启动的，相同名字，只能启动一个，而且名字不能动态变化

Posted by: 聪聪 | (4) April 15, 2014 08:32 PM

@zcpro

skynet 里不直接使用系统 fd 。

skynet socket api 自己分配一套 id ，是自增不重复的。

Posted by: Cloud | (3) April 15, 2014 04:23 PM

发送优先级很受启发. 不过如果不是那种特别实时的数据, 可以考虑用web的方式传送呀.

Posted by: cat | (2) April 15, 2014 04:16 PM

"直接把 socket fd 交给业务处理的服务，它直接向 socket 发包"--直接使用fd很容易产生问题，因为linux会重用fd，close一个socket，再新上来一个连接，有可能fd和刚关闭的那个相同，以前被这个问题坑过。

Posted by: zcpro | (1) April 15, 2014 03:29 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