云风的 BLOG
思绪来得快去得也快，偶尔会在这里停留

« 一个游戏的想法 | 返回首页 | skynet 消息服务器支持 »

计划给 skynet 增加短连接的支持

不基于一个稳定 TCP 连接的做法，在 web game 中很常见。这种做法多基于 http 协议、以适合在浏览器中应用。

运行在移动网络上的游戏，网络条件比传统网络游戏差的多。玩家更可能在游戏进行中突然连接断开而导致非自愿的登出游戏。前段时间，我实现了一个库来帮助缓解这个问题 。

如果业务逻辑基于短连接来实现，那么也就不必这么麻烦。但是缺点也是很明显的：

每对请求回应都是独立的，所以请求的次序是不保证的。

服务器向客户端推送变得很麻烦，往往需要客户端定期提起请求。

安全更难保证，往往需要用一个 session 串来鉴别身份，如果信道不加密，很容易被窃取。

即使有这些缺点，这种模式也被广泛使用。我打算在下个版本的 skynet 中提供一些支持。

所谓支持、想解决的核心问题其实是上述的第三点：身份验证问题；同时希望把复杂的登陆认证，以及在线状态管理模块可以更干净的实现出来。

我不打算基于 HTTP 协议来做，因为有专有客户端时，不必再使用浏览器协议。出于性能考虑，建立了一个 TCP 连接后，也可以在上面发送多个请求。仅在连接状态不健康时，才建议新建一个 TCP 连接。

由于不依赖长连接，所以登陆和游戏业务可以分到独立的地方去做。

整体系统有这样四类实体：登陆服务器、认证服务器、游戏服务器、客户端。

客户端进入游戏的流程是这样的：

1. 客户端先去认证服务器获得令牌。
2. 客户端把令牌交给登陆服务器，换取一个秘密。
3. 客户端利用获取的秘密，和游戏服务器建立加密通讯通道，进行游戏。

对于第一环节、通常是在第三方平台进行的（对于手游、通常是通过接入第三方平台的 SDK 完成）。这个令牌里通常包含用户身份 ID ，和用以校验令牌有效性的数据。

第二环节的登陆服务器，可以帮我们做系统的过载保护（当玩家人太多时，可以排队）。把登陆服务器独立出来，也方便不同的项目共享，不必每个项目都实现一次。它主要的工作是维持用户的在线状态，至于当用户已经在登陆状态时，再获取登陆请求时的处理：是顶掉老号，还是拒绝登陆，或是允许同时登陆，都放在第三环节中进行。

由于游戏服务器的登陆点可以有多个，我倾向于由客户端事先选好他的登陆点，然后把登陆点和第一环节得到的令牌交给登陆服务器。登陆服务器拿到令牌后，从中提取出用户唯一 id 并校验令牌是否有效（可以是本地校验，也可以是去第三方平台校验，这取决于第三方平台的协议）。

登陆服务器保持着用户在线状态，从设计上是允许同一用户有多个在线实体的（虽然不一定用的上），这可以仿造 XMPP 协议，为每个有效令牌的持有者分配一个唯一 id ：uid@登陆点/subid 。subid 是随机生成，且不重复的，当单个登陆允许多重登陆，subid 有实质意义，而在频繁登陆时，处理一些边界情况时也能发挥作用；uid 可以直接是第三方平台分配的 id ，也可以是登陆服务器自己生成的。自己生成有利于同时接入多个第三方平台。

登陆服务器可以接受在线状态查询，通过 id 可以查到该所有在线的完整关联 id 。

一旦用户完成登陆，登陆服务器就认为此玩家在线，再次收到同一玩家的登陆请求时，应该向所有已经在线的关联 id 所在的游戏服务器发送 RPC 查询请求（由于关联 id 中包含有登陆点信息，所以它知道该去哪里查询）。由游戏服务器来决定是否拒绝这次登陆还是踢掉前一个登陆，接受新的；当然也可能是玩家已经刚刚离线。

如果玩家主动离线、游戏服务器应该向登陆服务器发送 RPC 请求，注销在线状态。

客户端和登陆服务器的交互应该使用加密信道，如果不想使用标准 https 协议，也可以做简单的 DH 密钥交换，加上 RC4 XOR 加密信息流。

玩家登陆成功后，除了收到他所属的唯一 id 串外，还要接受一个秘密，用于和游戏服务器通讯。这个秘密事先在登陆服务器和游戏服务器交互时分发到游戏服务器了。游戏服务器也会提前做好准备某个特定 id 即将登陆。

在第三环节，客户端和游戏服务器握手时，先明文发送 uid@node/subid:id:randomkey 表明自己是谁，这里 id 表明是第几次握手；如果要重新建立连接，握手时 id 必须比之前的大，用过的 id 都会被服务器拒绝。

之后的信息都用 secret+id 来密钥来加密。randomkey 用于握手，游戏服务器在加密流的开头，先回应这个 randomkey ，如果用不匹配的密钥，会被游戏服务器检查出来断开连接。

如果客户端想重复和游戏服务器建立连接，它不需要再次去登陆服务器登陆。只需要把上次的 id 递增，并重新生成一个 randomkey ，去和游戏服务器握手即可。游戏服务器可以自己限制同一个用户能够同时建立通讯连接的上限，以节约服务器资源。

云风 提交于 July 11, 2014 06:13 PM | 固定链接

COMMENTS

短连接设计的再好是不是也避免不了没有网络。如果客户端在没网的时候也能继续玩，重新连接上的时候把游戏结果发到服务器比较好，服务器做校验而已。

Posted by: carp | (11) September 2, 2014 08:41 PM

如何及时有效的判断客户端已经离线？游戏服务器设定一个时长（例如5分钟），超过该时长客户端还没有请求就认为离线，然后向登录服务器注销该客户端的状态。

Posted by: niuliugou | (10) July 21, 2014 11:58 AM

如何及时发现网络已断开？我们现在心跳包都缩短到了3秒

Posted by: Acai | (9) July 16, 2014 11:17 PM

这是一个非常实用的功能

Posted by: terry8210 | (8) July 14, 2014 01:50 PM

TCP的友好，在传输大文件数据流时，意思就是“我会检测网速，如果网速慢就慢点传，大家都悠着点吧”

但这种友好对于传输游戏逻辑数据，一点意义都没有。游戏逻辑占用的带宽本来很少，程序员都会假设网络底层带宽远大于所需带宽。

对游戏而言，重要的是延时，而不是带宽。

TCP一丢包延时就暴增，完全不讲道理。难道还真怕玩家刷个喇叭就把ISP带宽占完了？

Posted by: Atry | (7) July 12, 2014 06:36 PM

重复造轮子有意思吗？有意思！！！

Posted by: tim | (6) July 12, 2014 06:28 PM

TCP over TCP能不能提高性能，是技术问题。答案很明显。

我觉得选择底层提供短连接，然后把重发幂等性这些问题统统让上层去发愁，可能不是特别好的选择。

至于这样做好不好，这是道德问题，或者说价值判断。价值判断取决于立场。

以ISP的角度，用户用电驴BT创建成百上千个TCP连接，抢咱家带宽，真是该死，必须要封杀啊。

Posted by: Atry | (5) July 12, 2014 06:18 PM

重新实现 TCP 不是不可以提高性能，而是一个道德问题.

TCP 是一个友好的协议，基于所有采用 TCP 协议的人都守规矩,从而达到集体的最大收益.

用短连接或 UDP 模拟 TCP 协议是没有意义的.

Posted by: Cloud | (4) July 12, 2014 02:45 PM

既然选择短连接模型，业务层就要考虑"同一请求被重复发出" 是一种常态了.

Posted by: Cloud | (3) July 12, 2014 02:40 PM

虽然先前的文章你说“实现一个 TCP over TCP 是没有太大意义的”。但在中国国情下，自己实现TCP over TCP还真免不了。

Posted by: Atry | (2) July 12, 2014 10:45 AM

短连接并不能解决TCP不稳定的问题。

比如说客户端提交某个操作给服务端，结果在发请求的过程中，底层TCP不稳定，某个路由器丢包了。一般的TCP实现要连续丢3个包才会发RST，而丢3个包的总退避时间长达几十秒。就算连续丢包3次导致请求失败，客户端还不能重发请求，因为客户端无法判断服务器有没有处理前一个请求，随意重发会导致同一请求被处理两遍。

简单的说，用短连接会导致一下后果：

1. 当网络不稳定时，会卡几十秒。
2. 卡过几十秒后，就算网络恢复，客户端的数据与服务器会不同步（因为客户端不能假定先前请求成功还是失败），必须由上层逻辑执行断线重连流程重新与服务器同步所有数据。如果不执行断线重连逻辑，客户端数据就会错乱。

Posted by: Atry | (1) July 12, 2014 10:37 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