discussion01

1354202

1 低带宽、不稳定网络下保持长连接

问题：
与服务器的 TCP 连接经常莫名断开，客户端又检测不到，不能及时重连。

分析：
提取本机 TCP 状态检测是否连接正常，若不正常则使用心跳机制（本项目由服务端发送心跳包客户端接手后返回数据包--解决了部分断连问题）

方案：
1 是在应用层制定协议，发心跳包，这也是 C，JAVA 等高级语言比较常用的方法。客户端和服务端制定一个通讯协议，每隔一定时间（一般 15 秒左右），由一方发起，向对方发送协议包；对方收到这个包后，按指定好的通讯协议回一个。若没收到回复，则判断网络出现问题，服务器可及时的断开连接，客户端也可以及时重连。

2 通过 TCP 协议层发送 KeepAlive 包。这个方法只需设置好使用的 TCP 的 KeepAlive 项就好，其他的操作系统完成。操作系统会按时发送 KeepAlive 包，一发现网络异常，马上断开。这种方法也是有些缺陷：比如某一时刻，网线松了，如果刚好被 KeepAlive 包检测到，它会马上断开 TCP 连接。但其实这时候 TCP 连接也算是 established 的，只要网线再插好，这个连接还是可以正常工作的。一般的系统，如果没有设置 KeepAlive 值，默认是 2 个小时--可能难以保证 2 个小时内没流量而不断开连接，并且无法解决恶意连接的问题。

3 当网络异常时，客户端断开连接后还可以怎么知道？--找到所有本地 TCP 连接的信息，筛选出我需要的那个 TCP。查看本机所有 TCP 连接信息，网上一般的方法，都是通过程序调用 CMD 命令里的 netstat 进行，然后再分析其内容。但在 CMD 窗口显示完所有 TCP 信息需要 15s，或者更长时间，这在 chat 程序中是不能忍受的。还有方法：使用 iphlpapi.dll。这是一个在 win98 以上操作系统目录 System32 都包含的库函数--但是如何使用好像资料比较少。

关于 keepAlive：
主要为服务器应用提供，服务器应用希望知道客户主机是否崩溃，从而可以代表客户使用资源。如果客户已经消失，使得服务器上保留一个半开放的连接，而服务器又在等待来自客户端的数据，则服务器将应远等待客户端的数据，保活功能就是试图在服务器端检测到这种半开放的连接。

如果一个给定的连接在两小时内没有任何的动作，则服务器就向客户发一个探测报文段，客户主机必须处于以下 4 个状态之一：

1. 客户主机依然正常运行，并从服务器可达。客户的 TCP 响应正常，而服务器也知道对方是正常的，服务器在两小时后将保活定时器复位。

2. 客户主机已经崩溃，并且关闭或者正在重新启动。在任何一种情况下，客户的 TCP 都没有响应。服务端将不能收到对探测的响应，并在 75 秒后超时。服务器总共发送 10 个这样的探测 ，每个间隔 75 秒。如果服务器没有收到一个响应，它就认为客户主机已经关闭并终止连接。

3. 客户主机崩溃并已经重新启动。服务器将收到一个对其保活探测的响应，这个响应是一个复位，使得服务器终止这个连接。

4. 客户机正常运行，但是服务器不可达，这种情况与 2 类似，TCP 能发现的就是没有收到探查的响应。

题外：由此看出 KeepAlive 也不适合本项目。TCP 里同时有短连接机制，相对好实现，但并不适合 chat 程序--TCP 建立花销大，释放也大--握手机制。在看到的资料里 socket 通信还有关于 WCF 框架，和长连接问题并无关系，但是可以绑定 TCP 当客户端和服务端同时支持时也有一定优势--可靠。在看资料时还看到关于全球移动端到端的 WebRTC 方案--但是解决的是 QoE 的问题。

2 消息不重复不遗漏

问题：

重发机制会带来重复。遗漏--可能消息同时到达，网络不稳定造成中途丢失等等。

方案：

对每条接收的消息匹配 id，对不同 id 进行回复。但是当消息太多或者有恶意连接时，很消耗资源--在相对大规模时会出现问题。可以进行限制 ip 客户端发送消息上限和发送频率。并且对 id 进行循环式（类似滑动窗口）使用非线性增大。

3 压缩

方案：

关键在于序列化协议的选择（并需要考虑编解码的时间成本）。在 chat 类点对点程序里每次相对发送消息小但是频度高，如果协议占比太大那么将导致信道利用率降低。netty 的编解码速度相对快，且 google 的 proto buffer 相对性能较高。

关于压缩有很多可应用算法/不同编码--涉及底层。

1.Google 最近发明的 HTTP 压缩算法 SDCH

2.HTTP 中的压缩算法 Gzip

3.设计模式之单例模式、享元模式。。。paper 里很多。