软件复用
讨论课文档

1352920 冯夏令

复用解决方案

1.长连接心跳机制

为了解决用户始终在线的问题，服务端需要跟客户端保持一个长连接，在我们的项目中，我们使用了基于Socket的TCP连接。理论上来说，TCP连接是一直保持着连接状态的，但是在实际情况中，如果网络中的结点发生了故障，或者该连接因在一定时间内没有数据交互而被防火墙自动断开的时候，该连接就会中断，这个时候就需要心跳机制，用于确认网络的连接状态，保证连接的有效性。

心跳机制是这样在我们的项目中运用的：服务端每隔一段时间向客户端发送一个数据包（也叫心跳包），客户端在收到服务端的心跳包后，给服务端发回一个反馈数据包，表示该连接是有效的；如果在一定时间内服务端没有收到客户端发回的数据包，则服务端认为该连接已经中断，需要重新进行连接。

保持长连接的操作过程如下：

2.消息不遗漏/不重复

长连接只解决了用户始终在线的问题，而在网络不稳定的情况下，客户端服务端直接发送的消息很可能发送遗漏，简单的处理消息遗漏的办法是给每一条消息指定一个唯一的序号，如果接收方收到的消息遗漏了一个编号，那么说明遗漏的编号对应的消息需要发送方重新发送该条消息。

在我们的项目中，我们是这么解决消息不遗漏/不重复功能的：客户端和服务端各维护一个消息队列，在需要发送消息的时候，发送方将需要发送的消息编上一个唯一的序号并加入到发送方的队列中，并从队头一个个取出消息，并记录下发送的时间。接着将发送过的消息重新加入到队尾中，如果该消息还没重新回到队头中并且已经接收到接收方发送的Ack包时，将该消息从队列中移除；如果该消息重新回到队头并且没有收到接收方发送的Ack包时，先检查是否发送超时（使用当前时间减去上次发送时间，再与定义好的TimeOut进行比较），如果超时则重新发送，如果未超时则等待。

该过程如下图所示：

3.消息压缩

在低带宽的网络环境中，如果发送的消息数据量过大，则会导致消息传输过慢甚至无法传输的问题，在这种情况下我们考虑消息的压缩。
很遗憾我们的项目中没有实现消息的压缩，在后续的版本中我们可以把该功能加入。我的考虑是使用目前广泛使用的文件压缩技术GZIP。使用GZIP的其中一个原因是因为GZIP在取得很好的压缩比例（20%）的同时，不会占用太多的CPU资源。