Problem 1

1352978 施闻轩

需求

用户登录后始终在线，考虑低带宽/不稳定网络

具体问题

1. 保持长连接

由于大多数防火墙和路由器会静默地抛弃长时间没有活动的 TCP 连接（此操作不会触发 C/S 端的 Socket#Close 事件），因此需要使用心跳机制保持连接活动。

服务端可以每隔一段时间向客户端发送一个心跳包，保持服务端到客户端信道的活动性（并及时检测到中断的连接），客户端收到心跳包后则反馈数据包，保持客户端到服务端信号的活动性（同样可以检测到中断的连接）。

在本项目中，这是已经实现的功能，服务端 `HeartbeatHandler` 负责在信道空闲时发送心跳包，客户端 `HeartbeatPushHandler` 负责在收到服务端心跳包时发送反向数据包。心跳包间隔目前是 30 秒。

另外，客户端应当在超过一定时限没有收到心跳包或其他数据包的情况下关闭连接，并尝试重连。这在目前的代码中没有实现。

Linux / Windows 的 Socket 接口中均支持 Keep Alive 参数，同样使用心跳包机制实现长连接。但 Keep Alive 对开发者是透明的，开发者无法控制重试和心跳逻辑，也无法获知重试和心跳，因此在本项目中没有被采用作为唯一的保持长连接方式。

2. 消息不遗漏、消息不重复

对于一般应用，为了确保消息（指服务端和客户端之间一个完整的帧）不遗漏、不重复，可以给每一个消息指派唯一编码，以消息为粒度进行可靠性控制。例如，强制要求每一个消息都需要确认 (acknowledge)，这样在没有收到消息对应的 ack 时可以对消息进行重发。由于每个消息都有唯一编码，因此可以避免接收端收到重复消息。本项目中虽然也使用了消息序列号和 ack 机制，但由于项目需求中没有不可靠信道的需求，因此仅仅用这些机制实现了 Request - Response，而没有实现消息不遗漏和重发。

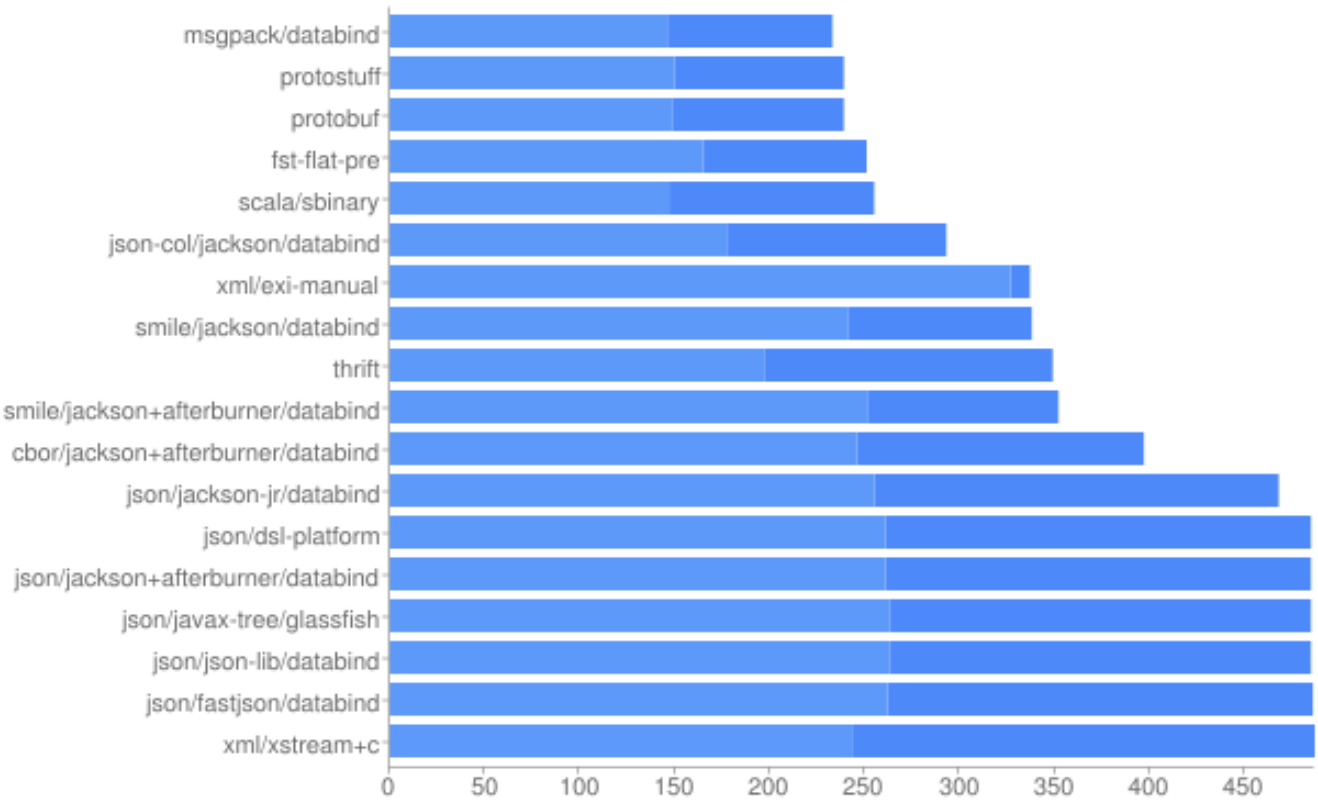
特别地，对于即时通信类或数据推送类等经常涉及一批消息超递的应用，可以使用同步方式解决消息遗漏

特别地，对于即时通信类或数据推送类等经常涉及一批消息投递的应用，可以使用同步方式解决消息遗漏和重复问题，即服务端与客户端每次都进行的是状态同步，如“上次同步时消息接收到这儿了，请发送给我从那以后所有的消息，作为这次同步的内容”。这是以一批消息为单位的，相比每条消息确认来说降低了网络负载，更重要的是它可以确保消息发送和接收有序性。

3. 消息压缩

本项目中，客户端和服务端之间的数据主要包含两部分：序列化载荷 (Serializer Payload)、消息载荷 (Message Payload)。

为了优化序列化部分的载荷大小，可以使用紧凑的基于二进制的序列化协议（如 Proto Buffer）替代常用的 JSON、SOAP(XML) 等序列化协议。以下图表来自 jvm-serializers，展示了常见序列化载荷大小。



本项目中考虑到 Netty 自带了 Proto Buffer 序列化中间件，且 Proto Buffer 具有显著的性能优势，因此直接使用了 Proto Buffer 作为序列化方式。从上图可见，使用 Proto Buffer 相比 JSON / XML 来说可以减少将近一半的体积。

对于消息载荷，即所被序列化对象中的数据，可以采用常见的 gzip 进行一定程度上的压缩。对于文本数据，使用 gzip 压缩一般可达到 20% 左右的压缩比。考虑到本项目是一个聊天室项目，用户不会无聊地发送大段内容（即使发送大段内容也不会达到 KB 级别），因此没有对消息载荷使用 gzip 进行压缩。

4. 低延迟和负载

在不稳定网络中（如移动设备），经常会出现数据包丢失、连接中断、延迟很大等情况，而 TCP 协议并不适合在这种不稳定信道下工作：在 TCP 中，一旦连接中断了就需要重新建立连接，而在应用中一般还

不适合在这种网络环境下工作。在 TCP 中，一旦连接中断了就需要重新建立连接，而在应用中一般还需要重新进行认证，这些额外的连接重新建立、初始化、认证等步骤显著地增加了负载，使得原本糟糕的情况变得更糟糕。因此，若主要用户都在此类网络环境下，适合使用 UDP 协议传输数据，载荷则以上述第二条描述的“同步”机制为主体，所有操作均在进行服务端和客户端状态的同步。