FastIM 框架设计文档

2014-6-21

Michael Tan

本文档描述FastIM项目整体框架设计，针对整个FastIM代码实现过程起指导作用，该设计方案应该与代码实现保持一致。

# 顶层设计

## 全局的部署视图

主要描述用户与服务器的位置关系以及服务性质



## 全局消息的交互协作视图

主要描述客户端与服务器的协作角色、交互情况、消息流向，定义了基本功能。

面对客户端和服务端的IO层时，每一条消息都通过服务器转发，到达目标客户端。



面对客户端平台逻辑层时，要做到的效果是——服务端透明，即客户端的逻辑层按业务封闭数据以后扔给客户端的IO层，由IO层保证正确与服务通信并反馈发送结果。



# 服务器框架设计

## 服务器的用例视图

主要描述服务器需要提供的功能用例



## 服务器实现的构件视图

主要描述通用服务器框架设计，实现层次的划分思想。



LogSystem——日志系统，分三个模块，分别记录IO层、业务逻辑层、数据层的日志；日志提供接口层以方便日志记录功能的改动，应该设计成另外的单独进程，使用管道传输日志信息。

IOLayer——网络接口层，主要负责处理客户端的连接请求和数据双向转输；使用多路分发的Reactor模型来处理，目前打算用成熟的LibEvent库来封装实现。这一层需要设计成完全不处理业务逻辑的功能；在转向业务逻辑层时加适配层，方便记录流量。

LogicalLayer——业务逻辑层，主要负责即时通讯相关的业务处理。业务逻辑层以核心功能代码为主，需要使用IOLayer的发送数据接口以及DataLayer的数据模型接口，但不应该包含直接的网络接口操作或数据层相关的操作——这一层不需要知道IOLayer的数据是怎么来的和怎么去的，也不需要知道DataLayer后面是什么数据库，甚至没有数据库也可以。由于此层业务较多，后续需要再划分细化功能。

DataLayer——数据层，主要负责数据库存储相关的操作，数据层的设计依赖业务逻辑层，为业务逻辑层提供数据模型接口，接口应分层，内层是最小化接口，外层封闭以后是适用业务接口。

## 线程框架设计

LogSystem单独进程实现，不在主框架线程设计范围内。

全局使用线程方案可以考虑以下两种方案之一：

### 客户连接线程一比一方案

这种方案逻辑相对清晰，一个客户对应一个线程；但这种设计方式不好的地方是如果过多用户持续连接的时候，为维护客户连接会消耗过多的线程资源。

1. 连接请求线程：此线程只负责客户的连接请求，不负责任何业务逻辑处理；接收请求以后，启动新线程处理此连接的所有活动。全局只一个这样的线程。
2. 客户连接线程：当连接请求到达的时候创建此线程；此线程接收连接上的数据，并完成发送数据；对每个客户维护这样的一个线程，负责维护客户行为状态机数据；有多个少个客户处于连接状态，就有多少个客户连接线程。如果不需要跟其他客户连接线程交换信息则不需要与业务处理线程协调处理，如果有需要与其他客户连接线程协调处理，则把任务投递到业务处理线程——比如发送一个消息给好友。
3. 业务处理线程：此线程处理客户连接线程递交的请求，如转发消息，发起会话等。
4. 数据模块线程：因为数据读写处理操作大多数是IO行为，可以集中管理，只提供一个处理线程，外部请求线程(主要是客户连接线程)以同步形式投递请求到线程的任务队列后返回，数据模块线程从任务队列中获取任务处理完请求以后，同步投递数据结果给外部请求线程后返回。

对应客户连接线程需要维护的状态机方案：

1. TCP已连接(需要设计超时)
2. 用户已登入
3. 用户已登出
4. TCP已断开

### 半同步半反应堆线程池方案

此方案要求所有客户请求都是无状态的，因为同一个连接上的不同请求可能会由不同的线程处理。如果不能做到无状态，可以考虑使用额外的数据管理调度关系。

1. 连接请求线程：此线程只负责客户的连接请求，不负责任何业务逻辑处理；收到连接请求以后，记录足够的信息后创建对应连接请求任务后插入连接请求任务队列，返回继续处理其他连接请求。
2. 连接处理线程：此线程维护连接请求队列，循环从连接请求队列中取出连接请求，创建连接信息结构并维护此结构，并且继续监听所有已连接客户的数据到达，数据到达以后使用对应的连接信息结构数据，交给线程池中的一个空闲线程处理。
3. 线程池中的处理线程：此线程池中任何线程都可以根据传入的连接信息结构处理任何连接业务逻辑，包括解析请求包，处理请求，处理数据，响应请求等。

# 实现设计

## 服务器各组件的状态机视图

## 服务器各组件之间的交互视图

## 消息数据的活动视图

## 类的静态视图

# 代码设计