

**2015 届本科生 毕业 论文**

**基于TCP和Websocket协议的即时通信系统**

学 号： 201101014006

姓 名： 陈建华

学 院： 计算机学院

指导老师： 朱懿敏

专 业： 计算机科学与技术

完成时间： 2015年4月 15日

**学位论文独创性声明**

本人郑重申明：

1. 本人所呈交的 学 士学位论文《基于TCP和Websocket协议的即时通信系统》是我个人在导师指导下进行的研究工作及取得的研究成果。
2. 除了文中特别加以标注和致谢的地方外，论文中不包含其他人已经发表或撰写过的研究成果。
3. 与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确的说明并表示了谢意。

学位论文作者（签名）：

年 月 日

# 摘　　要

随着网络通信和计算机技术的发展，人们越来越希望能够即时发送和接收互联网消息。与此同时随着互联网的发展在HTML5中提出了websocket协议，能更好的节省服务器资源和带宽并且服务器和浏览器能够双向实时通讯。为了能让用户体验传统客户端和web带来的即时通信结合的超爽体验，本次毕业设计将完成web端和客户端之间的即时通信系统，并利用在大学所学的知识打造高性能，高并发，高可拓展性服务器。同时更是为了方便人们之间的信息交流，让用户随时的接入即时通信，提升人们生活的效率。

该按照软件工程思想采用面向对象和模块化设计方法设计。其中采用了网络通信技术、数据库技术，多线程技术和信息安全技术的基础上设计并实现新型即时通信系统。该系统为用户提供点对点私聊，群聊，离线消息，消息提示于一体的即时通信平台。

**关键字：**即时通信；高性能；高并发；TCP/IP；Websocket

# ABSTRACT

With the development of network communication and computer technology, more and more people want to be able to send and receive instant messages on the Internet. At the same time with the development of the Internet in HTML5 websocket Protocol is proposed, to better conserve server resources and bandwidth and the server and the browser is capable of two-way communication in real time. In order to make the user experience of traditional Web and client of instant messaging with super-cool experience, this design will complete instant communications between the Web server and client systems, and use the knowledge learnt at University to create a high performance, high concurrency, highly scalable server. But in order to facilitate the exchange of information between people, allowing users to access instant messaging at any time, make people's lives more efficient.

According to the software engineering design method of object-oriented and modular design. Which uses the network communications, database technology, multi-thread technology and information security technology designed and implemented on the basis of a new instant messaging system. The system provides a point-to-point private chat, group chat, offline messages, a message in one of the instant messaging platform.

**Keywords:** Instant Messaging; high-performance, high concurrency; TCP/IP;Websocket

**目录**

[摘　　要 III](#_Toc416861726)

[ABSTRACT IV](#_Toc416861727)

[第一章 引言 1](#_Toc416861728)

[1.1 即时通信系统基本概念 1](#_Toc416861729)

[1.2 即时通信系统的发展历程 1](#_Toc416861730)

[1.3 系统研究目的和意义 1](#_Toc416861731)

[1.4 系统可行性分析 2](#_Toc416861732)

[第二章 相关技术介绍 3](#_Toc416861733)

[2.1 TCP/UDP协议 3](#_Toc416861734)

[2.2 WebSocket协议 3](#_Toc416861735)

[2.3 服务器模型 4](#_Toc416861736)

[2.3.1 同步 4](#_Toc416861737)

[2.3.2 复制进程 4](#_Toc416861738)

[2.3.3多线程 5](#_Toc416861739)

[2.3.4 事件驱动 5](#_Toc416861740)

[2.4 TLS/SSL协议 5](#_Toc416861741)

[2.5编程语言 5](#_Toc416861742)

[2.5.1 C++ 5](#_Toc416861743)

[2.5.2 Nodejs 6](#_Toc416861744)

[2.6 开源库 6](#_Toc416861745)

[2.6.1 Boost C++ Libraries 6](#_Toc416861746)

[2.6.2 Openssl 7](#_Toc416861747)

[2.6.3 Protobuf 7](#_Toc416861748)

[2.6.4 Hiredis 7](#_Toc416861749)

[2.6.5 Socket.io 7](#_Toc416861750)

[2.7 开发相关工具 8](#_Toc416861751)

[2.7.1 Redis 8](#_Toc416861752)

[2.7.2 Sqlite 8](#_Toc416861753)

[2.7.3 Nginx 8](#_Toc416861754)

[2.7.4 Visual Studio 2013 8](#_Toc416861755)

[2.7.5 Visio 2013 9](#_Toc416861756)

[第三章 即时通信系统的设计 10](#_Toc416861757)

[3.1 即时通信系统功能描述 10](#_Toc416861758)

[3.2服务器设计 11](#_Toc416861759)

[3.3数据库设计 11](#_Toc416861760)

[3.4主要的消息时序图 12](#_Toc416861761)

[3.4.1 未读消息时序图 12](#_Toc416861762)

[3.4.2点对点私聊时序图 13](#_Toc416861763)

[3.4.3群聊消息时序图 14](#_Toc416861764)

[3.5网络处理设计 15](#_Toc416861765)

[3.5.1封包 15](#_Toc416861766)

[3.5.2拆包 16](#_Toc416861767)

[3.6负载均衡设计 17](#_Toc416861768)

[3.7网络安全设计 18](#_Toc416861769)

[第四章 即时通信系统的实现 19](#_Toc416861770)

[4.1主要类和文件说明 19](#_Toc416861771)

[4.2系统部分截图 23](#_Toc416861772)

[第五章 即时通信系统的测试 26](#_Toc416861773)

[5.1测试目的和任务 26](#_Toc416861774)

[5.2测试流程 26](#_Toc416861775)

[5.3 功能测试用例 27](#_Toc416861776)

[第六章 结论和展望 28](#_Toc416861777)

[6.1 结论 28](#_Toc416861778)

[6.2 进一步工作的方向 28](#_Toc416861779)

[参考文献 29](#_Toc416861780)

[致谢 30](#_Toc416861781)

# 第一章 引言

## 1.1 即时通信系统基本概念

即时通信（Instant Messaging，缩写为 IM），是以互联网为基础，实现交互双方文字、图片、视频、语言等信息实时传输。即时通信系统的最大特点就是能在网络上进行信息之间的实时交流。与之间交流、电话交流等通信方式相比，即时通信具有迅速、方便、隐蔽性强等特点，在网络上可以超越身份、年龄、行业、地域等限制，达到人与人之间的交流零距离，实时通信成为互联网时代人与人沟通的重要方式[1]。

## 1.2 即时通信系统的发展历程

1970年代早期，一种最早的实时通信形式是柏拉图系统（PLATO system）。之后在1980年代，UNIX／Linux系统的实时通信被广泛的使用于军事界与学术界，随着互联网的普及即时通信迅速的运用在各行各业。1996年11月，首个广泛被非UNIX/Linux用户用于互联网的实时通信软件ICQ被推出来了，英文“I seek you”是ICQ的谐称[1]。

随着互联网发展，即时通信系统在中国发展是非常迅速的。从1999年2月第一次推即时通信软件OICQ，即时通信在中国的发展已有16年左右历史。经历萌芽阶段后，即时通信在我国已经进入了迅速发展阶段。目前主要的即时通信产品主要有腾讯QQ、淘宝旺旺、飞秋、微软MSN、网易泡泡、百度Hi等。随着科技的发展，即时通信系统的功能日益丰富，即时通信不再只是即时交流简单文本内容，它已经发展成集娱乐、交流、电子商务、办公合作和知识共享等为一体的综合信息平台。它以迅速、低成本和方便等特点被越来越被人们所接受，即时通信成为互联网发展的趋势[1]。

## 1.3 系统研究目的和意义

随着网络通信技术和计算机技术的发展，人们越来越希望能够快速得发送和接收互联网消息，服务器管理员也希望减轻服务器的负担。与此同时随着互联网的发展在HTML5中提出了websocket协议，能更好的节省服务器资源和带宽并且服务器和浏览器能够双向实时通讯。同时也是学习和实践网络编程、操作系统、软件工程、数据库概论、算法等相关知识。

为了能让用户体验传统客户端和web带来的即时通信结合的超爽体验，并利用在大学所学的知识打造高性能，高并发，高可拓展性服务器。同时更是为了方便人们之间的信息交流，让用户随时的接入即时通信，提升人们生活的效率。

## 1.4 系统可行性分析

可行性研究(feasibility study)的目的，是弄清楚将要开发的项目是不是能够实现和值得去实现，也是在高层次上进行的一次大大简化了的需求分析与设计。即使研究的结论是不值得进行，所花费的精力也并不白费，因为他避免了一次更大的浪费[2]。

对研究中可能提出的任何一种解决方法，都要从经济、技术、运行和法律等诸多方面来考虑其可行性，最终给出明确的结论来决策是否可行。

①经济可行性。即时通讯可以不赚钱，但没有却是不行的，就像一个地方要致富，不修路是不行的道理一样。有了大量用户群，提供其他服务自然会带来大量收益。因此经济上是可行的。

②技术可行性。通过TCP/UDP协议客户端的即时通信已经有很多成功的软件；在之前服务器要推送数据到浏览器，通过Ajax定时查询，严重浪费互联网和服务器资源。随着互联网的发展在HTML5中提出了websocket协议，能更好的节省服务器资源和带宽并达到实时通讯。通过消息队列实现不同类型服务器的互联互通。因此技术上是可行的。

③运行可行性。目前已经学习了C++语言和JavaScript语言，学习了网络编程、软件工程、操作系统、数据库原理等课程。因此运行是可行的。

④法律可行性。采用开源库openssl、protobuf、hiredis，开源数据库软件redis、代理服务器nginx。因此在法律上是可行的。

综合上述分析，新的系统是可行的。

# 第二章 相关技术介绍

## 2.1 TCP/UDP协议

TCP---传输控制协议,提供的是面向连接、可靠的字节流服务。在传输数据之前必须建立连接，数据传输结束后要释放连接。TCP提供超时重发，读取数据的次序一致，丢弃重复数据，检验数据，流量控制等功能，保证数据的可靠传输。

UDP---用户数据报协议，主要特点：无连接、尽最大努力交付、面向报文、没有拥塞控制、支持一对一、一对多、多对一、多对多的交互通信、头部开销较小。在传输数据之前不需要建立连接。原地主机的运输层在收到UDP数据报后，不需要给出任何确认。虽然UDP不提供可靠传输，但在某些情况下确实一种最有效的工作方式。

Socket是应用层与TCP/IP协议族通信的中间软件抽象层，它是一组接口。在设计模式中，Socket其实就是一个门面模式，它把复杂的TCP/IP协议族隐藏在Socket接口后面，对用户来说，一组简单的接口就是全部，让Socket去组织数据，以符合指定的协议[3]。

## 2.2 WebSocket协议

WebSocket是HTML5开始提供的一种在单个 TCP 连接上进行全双工通讯的协议。WebSocket通信协议于2011年被IETF定为标准RFC 6455，WebSocketAPI被W3C组织定为标准，能更好的节省服务器资源和带宽并且服务器和浏览器之间能够实时双向通信。

使用websocket的话，浏览器只需一个TCP连接即可完成双向通行，在服务器和浏览器频繁通信时，TCP连接可以得到高效应用，编程模型也十分简洁。Websocket协议主要分为两个部分：握手和数据传输。浏览器建立连接时，通过HTTP发起请求报文，与普通HTTP请求协议略有区别在于协议头upgrade：websocket；connection：upgrade；表示请求服务器升级协议为websocket。一旦websocket握手成功，当前连接将不在进行HTTP的交互，而是开始websocket的数据帧协议，实现浏览器和服务器的数据交互[4]。

 图2.1 websocket 协议升级示意图

## 2.3 服务器模型

### 2.3.1 同步

最早的服务器，其执行模型是同步的，它的服务模式是一次只为一个请求服务，所有的请求都需要按次序等待服务，这意味着除了当前请求被处理外，其他请求都处于等待的状态。

### 2.3.2 复制进程

为了解决同步架构的并发问题，一个简单的改进是通过进程的复制同时服务更多的请求和用户。这样每个连接都需要一个进程来服务，这是非常昂贵的代价。在进程复制的过程中，需要复制进程内部状态，同样的状态会在内存中存在很多份，造成内存浪费启动也是很缓慢。

### 2.3.3 多线程

为了解决进程复制中的浪费问题，多线程被引入服务器模型，让一个线程服务一个请求。线程对进程的开销要小许多，并且线程之间可以共享数据，内存浪费的问题可以得到解决，并且利用线程池可以减少创建和销毁线程的开销，但多线程所面临的并发问题只能说比多进程略好，因为每个线程都有自己独立的堆栈，这个堆栈都需要占用一定的内存空间，当线程数量过多时，时间将会被消耗在上下文切换中。

### 2.3.4 事件驱动

多线程的服务模型服役了很长一段时间，当并发增长到上万时，内存好用的问题将会暴露出来，这既是著名的C10K问题。为了解决高并发问题，事件驱动的服务模型出现了，避免了不必要内存开销和上下文切换开销。异步事件驱动模型中，把会导致阻塞的操作转化为一个异步操作，主线程负责发起这个异步操作，并处理这个异步操作的结果[4]。

## 2.4 TLS/SSL协议

SSL（Secure Sockets Layer，安全套接层），是Netscape开发出来的，用来保障网络传输数据时的安全性。TLS与SSL先在握手阶段采用RSA非对称，之后的数据传输在传输层对网络连接进行加密。TLS/SSL是个公钥/私钥的结构，它是一个非对称的结构，每个服务器和客户端都有自己的公私钥。它建立在可靠的TCP传输协议之上，为高层协议提供数据封装、压缩、加密等基本功能的支持,用于在真正的数据传输开始前，连接的双方进行身份认证、协商加密采用的算法、交换加密采用的密钥等[3]。

## 2.5编程语言

### 2.5.1 C++

C++起源与C语言，最初的名字为C with Class，意为带类的C语言。在《Effective C++》一书中的条款一中将C++视为一个语言联邦[6]。C++可以看成由四个部分组成：

1. C。C++以C为基础。预处理器(preprocessor)、数组(arrays)、指针(pointers)、自己分配内存、自己管理内存等都来自C。
2. 面向对象。类(class)、继承(inheritance)、封装(encapsulation)、多态(polymorphism)、虚函数(virtual function)等都是面向对象设计在C++上的最直接的体现。
3. 模板。C++的范型编程(generic programming)部分。他们带来新的编程范型(programming paradigm)，也就是所谓的template meta programming(TMP，模板元编程)。
4. STL。即template程序库，对容器、迭代器、算法以及函数对象的规约有极佳的紧密配合与协调。

C++11标准包括大量的新特性：包括右值引用、lambda表达式、常量表达式、类型推导关键字auto、decltype，函数因子、定则表达式，哈希表、多线程、随机数引擎等[7]。

### 2.5.2 Nodejs

Node.js是一个基于V8引擎的JavaScript运行时建立的平台，用于高效率地搭建快速响应、高性能、高并发的网络应用。Node.js 使用事件驱动，非阻塞异步的I/O 模型而得以轻量和高效，非常适合在分布式设备上运行的数据流动密集型的实时应用。Node.js采用C++语言编写而成，它不是Javascript应用，而是一个服务器端的Javascript的运行环境。[4]。



图2.2 浏览器与Node.js以及CommonJS组织、W3C组织、ECMAScript之间的关系

## 2.6 开源库

### 2.6.1 Boost C++ Libraries

Boost库是一个可移植、提供源代码的C++库，作为标准库的后备，是C++标准化进程的开发引擎之一。 Boost库由C++标准委员会库工作组成员发起，其中有些内容有望成为下一代C++标准库内容。在C++社区中影响甚大，是不折不扣的“准”标准库。Boost由于其对跨平台的强调，对标准C++的强调，与编写平台无关[8]。

### 2.6.2 Openssl

OpenSSL 是一个强大的开源SSL\TLS安全套接字层密码库，包括主要的密码算法（如：RSA，AES,DES）、生成数字证书封装管理功能及SSL协议的实现，并提供丰富的测试程序和示例程序。[10][11]。

### 2.6.3 Protobuf

Protocol Buffers 是一种轻便高效的结构化数据存储格式库，可以用于结构化数据序列化，可以根据预定义的消息类型生成高效的程序文件，很适合做性能要求很高的数据存储或数据交换格式。它广泛于自定义通讯协议、数据存储等领域的平台无关、语言无关、可扩展的序列化结构数据传输格式。目前提供了 C++、Python 、Nodejs、Java几种语言的接口[9]。

### 2.6.4 Hiredis

hiredis是redis数据库的C接口,原开源库只支持Linux。在通过宏判定修改hiredis的网络接口后能够支持windows[12]。异步连接通过adapter使用libevent。

### 2.6.5 Socket.io

Socket.io是一个开源的高效WebSocket库，它通过Node.js实现WebSocket服务端库，同时也提供客户端javascript库。Socket.io库支持以事件驱动为基础的双向实时通信，它兼容各种浏览器或移动设备。

Socket.io支持4种协议：Websocket、jsonp-polling 、xhr-polling、htmlfile，它会自动根据浏览器选择适合的连接方式，从而让开发者可以着重于功能逻辑而不是平台的兼容性问题 [13]。安装方式: npm install socket.io –g。

## 2.7 开发相关工具

### 2.7.1 Redis

Redis是一个key-value高速缓存数据库，它每秒操作次数能高达十万次。和Memcached缓存数据库类似，它支持存储的value数据类型非常丰富，包括list(链表)、string(二进制字符串)、set(集合)、zset(有序集合)和hash（哈希表类型），存储的数据可以以二进制数据存储。这些数据类型都支持push/pop、lrange、Sadd及取交集并集和差集及更丰富的数据操作，而且这些操作都是原子操作，也支持多操作的原子性操作。支持发布及订阅功能，为系统提供低耦合的交互模式。与memcached一样，为了保证高性能，数据库数据都是缓存在运行内存中。区别的是redis会周期性的把数据以快照的方式写入本地磁盘以实现持久化数据，并且在此基础上实现了master-slave(主从)同步[14]。

### 2.7.2 Sqlite

Sqlite，是一个轻量级的数据库，是遵守ACID的关系型数据库原理的系统，它以C语言实现非常高效。它是D.RichardHipp建立的开源项目。它的设计目标是能够嵌入到使用的软件中，而且目前已经广泛应用在很多软件产品，它占用内存资源非常的低，只需要占用几百K字节的内存[16][17]。

### 2.7.3 Nginx

Nginx ("engine x") 是一个著名的开源高性能的 HTTP 和反向代理服务器，它使用C语言开发网络库使用libevent性能高效代码结构优良，其源代码以类BSD许可证的形式发布。因它的高稳定性、高性能、丰富的操作功能、示例配置文件和低系统资源的占用而闻名，渐渐成为高性能网站使用比例最高的web服务器[15][18]。

### 2.7.4 Visual Studio 2013

Microsoft Visual Studio（简称VS）是美国微软公司强大的开发语言工具包系列产品。VS是一个功能丰富的开发工具，如代码提示、良好的错误提示、代码管控工具、UML工具、集成开发环境(IDE)等等。在最新的VS2013版本中支持了大部分C++11标准实现，新增了代码信息指示、Git版本控制支持还有单元测试支持等。

### 2.7.5 Visio 2013

Visio是一款需求分析、系统设计和流程进行可视化软件。对UML统一建模语言很好的支持，同时也包含有很多行业可视化工具，为人们更好的表达设计思路和想法，提高工作生活的效率。

# 第三章 即时通信系统的设计

## 3.1 即时通信系统功能描述

随着web的发展，在HTML5标准中提出了Websocket protocol协议。它实现了浏览器与服务器全双工通信(full-duplex)。为用户提供能够通过客户端和web端接入的即时通信平台，让用户体验传统客户端和HTML5标准结合带来的超爽体验，方便人们之间的信息交流，创建新型沟通文化，提升人们生活的效率。

该系统具有私聊，群聊，消息通知、离线消息存储，上下线通知等基本功能，同时保证数据通信的即时性以及通信过程和消息存储的安全性。



图3.1 用户用例图

## 3.2系统总体设计

TCP服务器和客户端采用C++编写，通过TCP协议与服务器连接并由SSL安全传输层保证安全，实现即时通信。浏览器端通过HTML5中提出的Websocket协议与服务器连接，Websocket服务器和HTTP服务器采用Nodejs编写。不同物理服务器，不同语言服务器采用统一的数据传输格式来传输，借助redis数据库的Pub/Sub模式和消息队列，来进行服务器之间的信息通信。对数据库进行Master/Salve备份。

 图3.2 服务器设计图

## 3.3数据库设计

为了提高对数据的处理，采用缓存数据库Redis。Redis是一个Key-Value数据库，数据库设计主要体现在Key的设计。存储的数据是二进制安全的。

哈希表user:(userid)存储用户信息。如user:14006。

表3.1 用户信息哈希表

|  |  |
| --- | --- |
| useId | 用户ID |
| usename | 用户名 |
| password | 密码 |
| imageurl | 头像 |
| email | 邮箱 |
| lastLoginTime | 最近登陆时间 |
| nickname | 昵称 |

集合groupSet:(userid)存储用户加入的群号。如groupSet:14006

集合friendsSet:(userid)存储好友ID。如friendsSet:14006

列表unreadMsg:(userid)存储用户二进制点对点未读消息。如unreadMsg:14006

键online:(userid)存储用户在线的服务器Socket信息。

哈希表group:(group)存储群信息。如group:1。

表3.2 群信息哈希表

|  |  |
| --- | --- |
| groupId | 群号 |
| groupName | 群名称 |
| imageUrl | 群头像 |

列表groupMsg:(groupid)存储二进制群消息。如groupMsg:1。

列表groupMember:(groupid)存储群成员ID。如:groupMember:1。

列表onlineGroup:(groupid)存储当前在线群成员ID。如：onlineGroup:1。

键unreadNumber:(userid):(groupid)存储未读群消息数目。如:unreadMember:14006:1。

哈希表loginMap:(userid)存储用户登陆密钥和消息服务器信息。如：loginMap:14006。

表3.3 用户登录信息哈希表

|  |  |
| --- | --- |
| address | 消息服务器地址 |
| port | 消息服务器端口 |
| sessionid | 密钥种子 |

有序集合serverRank存储消息服务器套接字信息及排名。

## 3.4主要的消息时序图

### 3.4.1 未读消息时序图

1，用户登陆后发出未读消息请求。

2， 查询unread列表所有私聊未读消息。如：LRANGE unread:14006 0 -1

3，向Response添加未读消息。

4，查询所有群未读消息。如：SORT groupSet:14006 GET # GET unreadNumber:14006:\*

5，向Response添加未读消息。

6，对Response序列化发送消息。

### 3.4.2点对点私聊时序图



图3.3 点对点聊天时序图

1, 一个用户发出一条点对点聊天消息。

2, 消息服务器接收到Peer消息，判断是否在当前服务器。

3, 如果接收的用户在当前服务器，向接收者的连接发送消息；如果用户不在当前服务器，查询用户状态。如：EXIST online:14006

4, 接收数据库查询结果，判断对方是否离线。

5, 如果对方离线，将二进制消息保存到对方未读消息列表；如果对方在线，根据返回结果得知对方Socket信息，向其Publish PeerMsg 二进制数据。

6, 其他服务器Publish 一条PeerMsg消息。

A, 订阅到一条PeerMsg:address:port消息，判断对方是否在服务器上。

B, 如果用户在服务器上，向其发送Peer消息；如果用户不在服务器，将二进制消息保存到对方未读消息列表。一般用户是会在服务器上的，防止发布消息的时候，用户退出的情况。

### 3.4.3群聊消息时序图



图3.4 群聊天时序图

1, 一个用户发出一条群聊天消息。

2, 消息服务器接收到一条群消息，向数据库查询群成员信息。如：SORT onlineGroup:1 by no\_exist\_key get # get online:\*

3, 获取到群当前在线人员信息。

4, 向本服务器的在线用户发送群消息；向其他服务器Publish包含用户ID列表和群消息的数据；将消息推入消息列表，如：LPUSH groupMsg:1 二进制数据；获取群成员和在线成员的差集，SDIFF groupMember:1 onlineGroup:1；更改未在线用户群消息未读数目，如INCR unreadNumber:14006:1；

A, 其他服务器Publish groupMsg消息包含用户ID列表和群消息。

B, 订阅到一条groupMsg消息，向用户ID列表中的用户发送群消息。

## 3.5网络处理设计

基于TCP协议为什么需要对数据进行封包和拆包：TCP是个面向可靠传输协议，就是没有边界的一串数据,就像河里面的水一样是连续的，其间没有分界线[22]。传输时出现粘包的原因有： 开启Nagle算法后，发送端需要等待缓冲区满才发送出去，造成粘包；接收方系统繁忙没有及时接收缓冲区的包，造成多个包连续等原因。假设我们连续调用两次send分别发送两段数据data1和data2,在接收端有可能出现以下几种接收情况：

A.先接收到data1,然后接收到data2。

B.先接收到data1的部分数据,然后接收到data1余下的部分以及data2的全部。

C.先接收到了data1的全部数据和data2的部分数据,然后接收到了data2的余下的数据。

D.一次性接收到了data1和data2的全部数据。

对于A这种情况正是我们需要的,对于B,C,D的情况就是大家经常说的"粘包",就需要我们把接收到的数据进行拆包,拆成一个个独立的数据包.为了拆包就必须在发送端进行封包。

对于UDP来说就不存在拆包的问题,因为UDP是用户数据报协议,也就是数据包是一个一个发,接收端不保证包的次序和是否到达，但能够确定一个UDP包是一个完整的消息。

### 3.5.1封包

封包就是给一段数据加上额外信息,使得在解析TCP数据时能够区分完整消息数据。由于消息是序列化成二进制数据进行传输，所以包需要附带消息的类型,过滤非法包时封包会加入"包尾"内容，如“\r\n”分隔符。



图3.6 消息封包图

### 3.5.2拆包

TCP每次读取到分隔符”\r\n”时作为一个包判断，读取头部网络字节序4 bytes数据,转换为本地字节序整数n，判断缓冲区大小是否等于n+4。如果相等则消费缓冲区n+4个字节;如果不相等则不消费缓冲区。

网络消息流程[20]



图3.7 消息流程图

1, 服务器异步读取数据，接收到一个TCP流后回调HandleRead函数。向缓冲区Commit接收到的字节数n。

2, 对缓冲区进行拆包，循环判断缓冲区是否有完整包，如果有调用ProtobufCodec的OnMessage函数，如果没有跳出循环。开始接收下一个TCP流。

3, OnMessage函数对缓冲区consume消费m个字节数，对二进制数据解码，生成对应的Message。

4, 调用ProtobufDispatcher的OnProtobufMessage函数将消息映射到服务器相应的处理函数。

5, 消息服务器对不同消息进行相应的处理。

6, 如果消息需要其他服务器处理则Publish相应服务器消息，如果用户在本服务器直接向其发送相应的消息。

7, 对消息进行封包，发送二进制数据。

## 3.6负载均衡设计

登陆服务器每次登陆后，取加权集合serverRank权值最大的服务器信息，对分配消息服务器的消息服务器进行加负权;用户退出后对分配的消息服务器加正权；

Web端通过配置Nginx来实现对Https和Websocket的负载均衡。

## 3.7网络安全设计



图3.8 网络安全连接图

1, 客户端登录时采用TLS/SSL连接，数字签名证书采用自签证证书，登录时传输密码的MD5校验码。

2, 登录服务器分配消息服务器和SessionID。

3, 连接消息服务器时第一个消息包作为握手包，不加密只包含用户ID。

4, 如果数据库有用户登录记录，返回成功。

5, 两端用SessionID作为种子对以后要传输的数据进行AES加密。

6, web端登录、拉取好友信息、群消息等用HTTPs（TCP短连接）。

7, 聊天消息采用在HTTPS建立的websockets连接（TCP长连接）。

# 第四章 即时通信系统的实现

## 4.1主要类和文件说明

Tcp\_Server:服务器网络连接类。采用Boost Asio作为异步模型，实现高性能网络服务器。监听端口、接收新的连接，对每个连接创建新的Tcp\_Session对象。

Tcp\_Session:网络连接Session类。接收、发送TCP流，PING数据包，错误处理，消息接收到后交付ProtubufCodec处理[22]。

Ssl\_Server:安全传输层TLS/SSL服务器网络连接类。

Ssl\_Session:安全传输层TLS/SSL连接Session类。

Connection\_Manager:网络连接管理类。管理连接用户ID与连接的映射关系，创建、退出时要处理的相关数据。

MyBuffer:数据缓存区类。

ProtubufCodec:网络消息封包和拆包及加密解密类。

成员变量存储ProtubufDispatcher的HandleMessage函数。



图4.1 ProtuCode类UML图

ProtubufDispatcher:处理函数和不同消息类型之间映射类。它有一个 map<Descriptor\*, ProtobufMessageCallback> 成员，通过registerMessageCallback 向Map表添加映射，ProtubufCodec解码后调用HandleMessage，找到回调函数，如果没找到调用默认回调。



图4.2 ProtubufDispatcher类UML图

Credis：与redis数据库操作类，像管道，多操作原子性等复杂操作通过返回redisContext来处理。

Server:服务器核心类，向ProtubufCodec,ProtubufDispatcher,tcp\_server或ssl\_server注册各种服务器处理回调，消息的对应处理方式，发布服务器间消息，订阅服务器间消息，操作数据库等操作，对于数据处理处理函数使用互斥锁确保在多线程下的安全。



图4.3 服务器启动和退出处理图

SessionID.h：密钥生成函数，通过当前系统时间戳和C++11的随机数生成引擎和正态分布生成15位长的密钥种子。

DemoServer.h:消息服务器启动文件，启动时接收服务器监听参数，创建和运行io\_service对象负责连接应用程序与操作系统的IO服务，控制台事件(如：Ctrl + c,关闭控制台)的监听以及处理。

LoginServer.h:登陆服务器启动文件。

Message.pb.h:登录时与登录服务器连接消息类集合。

MessageChat.h:登录后客户端和消息服务器消息类集合。

MessageServer.h:消息服务器之间的订阅消息类集合。

Node.js服务端

由于Node.js采用C++语言编写而成，它不是Javascript应用，而是一个Javascript的运行环境，基于libuv跨平台网络库，具有异步IO、事件驱动、单线程等特点。Websocket协议对TCP流封装成Frame数据包，websocket库采用Socket.io，数据传输采用JSON格式，TCP连接采用Node.js内置库net，Https也采用内置库https。

创建新的数据库用来对web端Session的高速存储，以用户名作为作为关键字，存储的内容是Session相关内容的JSON数据。包含SessionId、Session过期时间和最大生命周期。实现HTTP服务器重启后浏览器不需要重新登陆，关键字生存周期为一周，Session过期时间为30分钟，Session过期后更新数据库和浏览器Session的内容。

socket.js：websocket服务器文件，服务器启动和退出时流程与C++服务器相同，用于订阅发布消息、实时对聊天数据的处理、消息通知等实时通信操作。

multiProcess.js：由于Node.js是单进程单线程的，通过进程间通信传递相关句柄，实现Master-Worker模式主进程管理和调度工作进程的功能，充分利用多核CPU资源。



图4.4 Node.js服务器多进程示意图

HttpsServer.js：Https服务器文件，用户登陆、退出、拉取好友信息、群信息、分组信息、用户信息等操作。其HTTPS数据接口如表：4.1

表4.1 HTTPS数据接口

|  |  |
| --- | --- |
| /data/groupMember?groupId=1 | 群成员信息 |
| /data/friendsList | 用户好友列表 |
| /data/unreadPeerMsg | 所有好友未读消息 |
| data/groupsList | 群列表信息 |
| /data/unreadGroupMsg | 所有群未读消息 |



图4.5 HTTPS数据接口效果图

客户端和服务器很多文件公用:

DlgGroups.h、DlgFriends.h含有主窗口Tab页好友列表、群列表窗口对话框类。

DlgChat.h、DlgGroupChat.h含有点对点私聊、群聊天窗口对话框类。

ClientDlg.h、DlgLogin.h含有主窗口、登录窗口对话框类。

Web端

路由设计：/：首页；/user/userid：用户主页；/register：用户注册页面；/login：用户登录页面；/loginOut：用户退出页面；/Chat：聊天页面；/happyChat:无状态聊天页面，可以创建临时匿名会话。

client.js是websocket客户端的业务逻辑实现代码。

Ca.crt、server.crt、client.crt为ca证书、服务器证书、客户端证书；server.csr、client.csr为证书请求链，ca.key、server.key、client.key为密钥。

## 4.2系统部分截图

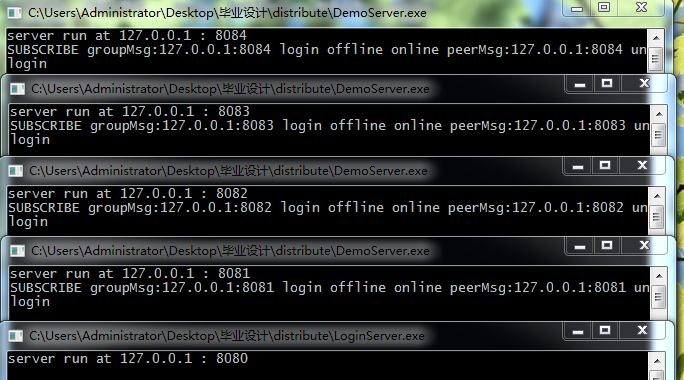


图4.6 C++服务器端启动控制台截图

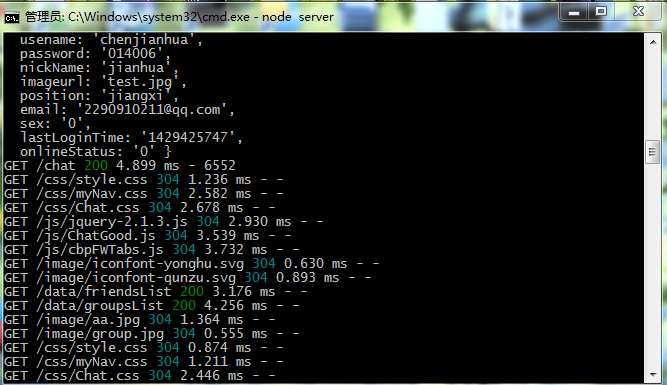


图4.7 Nodejs的HTTP服务器效果图

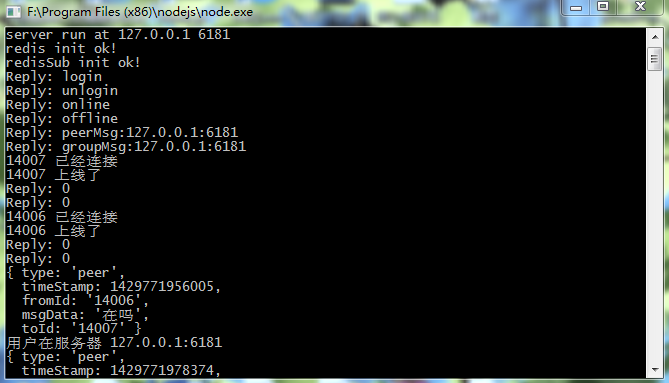


图4.8 Nodejs的Websocket服务器启动效果图

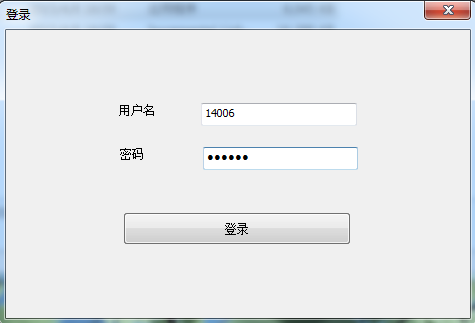


图4.9 C++登录界面

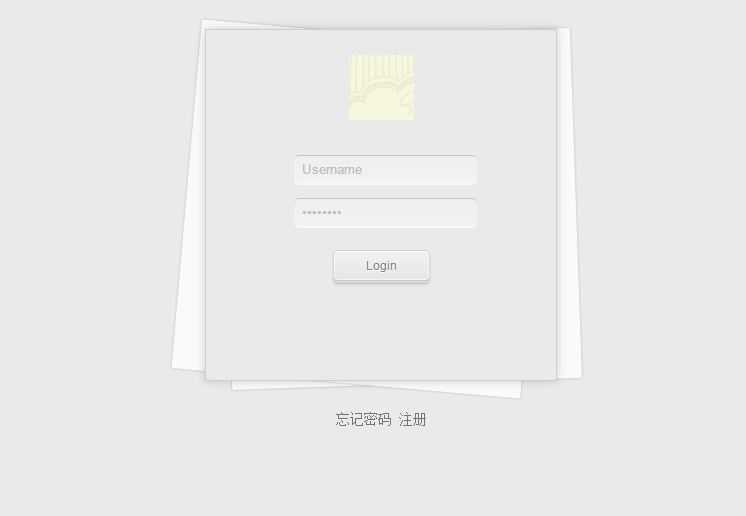


图4.10 web登录界面



图4.11 Web注册界面

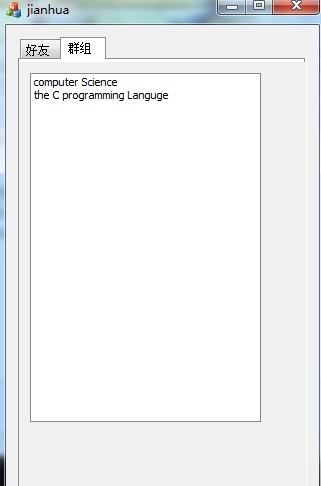
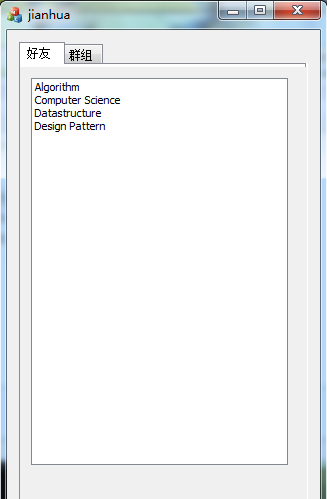


图4.12 客户端登录后界面



图4.13 Web端登录后界面

# 

# 第五章 即时通信系统的测试

## 5.1测试目的和任务

软件测试是对软件规格说明、软件设计和编码的最后复审，目的是在软件产品交互之前尽可能发现软件中潜在的错误。大量统计数据表明，软件测试成本往往占整个开发成本的1/3左右。随着人类对计算机的逐步深入，人们对软件的要求也越来越高，“不是人适应软件，而是软件适应人”等观点已经成为人们的共识。

测试是保证软件质量的重要手段，也是软件设计与开发过程中必不可少的环节。本系统在开发期间对每个功能都进行了单元测试，新的功能单元测试成功后才添加到系统都进行了集成测试，集成测试成功后才进行下一个功能开发，系统主要功能完成后对其进行了确认测试和系统测试。

即时通信系统的测试环境如下：

服务器5台：

CPU：Intel Core I7 2.8 GHz 8核心

内存：4.00GB

系统：Win7 64bits

缓存数据库：Redis 2.8.17

Node.js环境：0.10.33

客户端3台：

CPU：Intel Core I7 2.8 GHz 8核心

内存：4.00GB

系统：Win7 64bits

浏览器 ：Chrome、firefox

## 5.2测试流程

I，配置Nginx代理服务器实现对websocket的负载均衡；配置redis数据库实现Master/Slave备份和其他配置；npm install下载工程中package.json对应的Nodejs依赖库。

II，先4台开启C++消息服务器1台开启登录服务器，多个用户在不同机器上打开客户端之间并进行通信，对功能测试用例进行测试，观察服务器消息记录和消息流动。

III，开启Nodejs的HTTP服务器和Websocket服务器，多个用户在不同机器上的浏览器之间相互通信，对功能测试用例进行测试，观察服务器消息记录和消息流动。

IV，同时开启C++服务器和Nodejs服务器，在不同的机器上打开客户端和浏览器，两个用户之间一个使用客户端一个使用浏览器进行通信，对功能测试用例进行测试，观察不同服务器消息记录和消息流动。

## 5.3 功能测试用例

表5.1 功能测试用例

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| index | 功能模块 | 设计数据测试 | 预期结果 | 测试描述 |
| 001 | 用户注册 | 填写未注册的用户 | 注册成功 | 进入登录界面 |
| 002 | 用户注册 | 填写已注册用户 | 注册失败 | 提示用户已存在 |
| 003 | 用户注册 | 填写的信息不合法 | 注册失败 | 提示填写信息格式不合法 |
| 004 | 用户登录 | 用户ID和密码正确 | 登录成功 | 生成好友列表 |
| 005 | 用户登录 | 用户已登录 | 登录失败 | 提示已登录 |
| 006 | 用户登录 | 用户ID或密码错误 | 登录失败 | 提示用户ID或密码错误 |
| 007 | 添加好友 | 查找的用户ID存在且不是自己的好友 | 添加成功 | 刷新好友列表 |
| 008 | 添加好友 | 添加自己的用户ID | 添加失败 | 提示自己不能添加自己 |
| 009 | 添加好友 | 查找的用户ID已是用户好友 | 添加失败 | 提示已经加为好友 |
| 010 | 添加好友 | 查找未注册用户ID | 添加失败 | 提示找不到用户信息 |
| 011 | 发送消息 | 消息不为空 | 发送成功 | 服务器接收到消息 |
| 012 | 发送消息 | 消息为空 | 发送失败 | 提示消息不能为空 |

## 5.4 测试结果

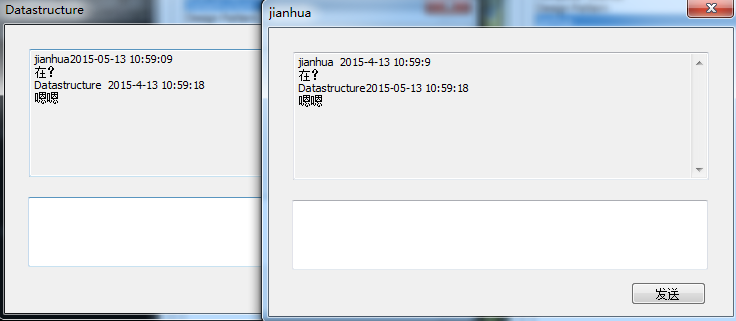


图5.1 客户端之间点对点私聊天

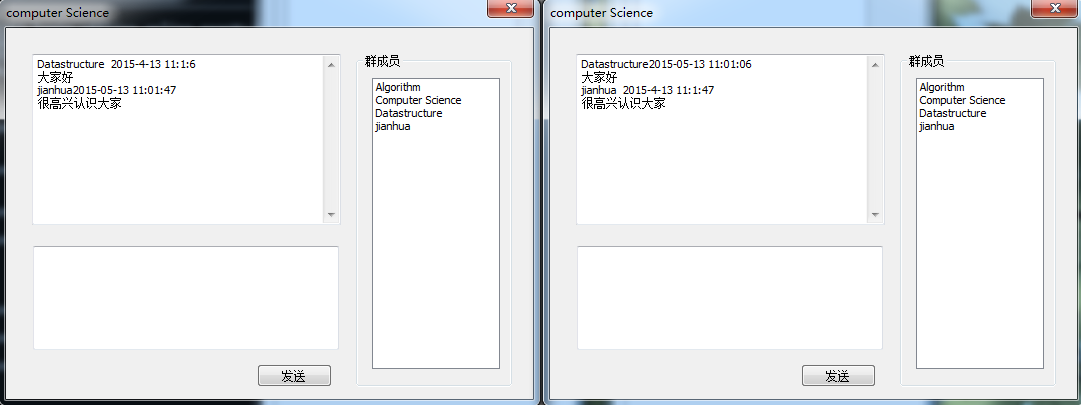


图5.2 客户端之间群聊天

两个用户登录后分别连接8083和8084端口服务器，二者相互聊天在服务器的信息,。如图5.3

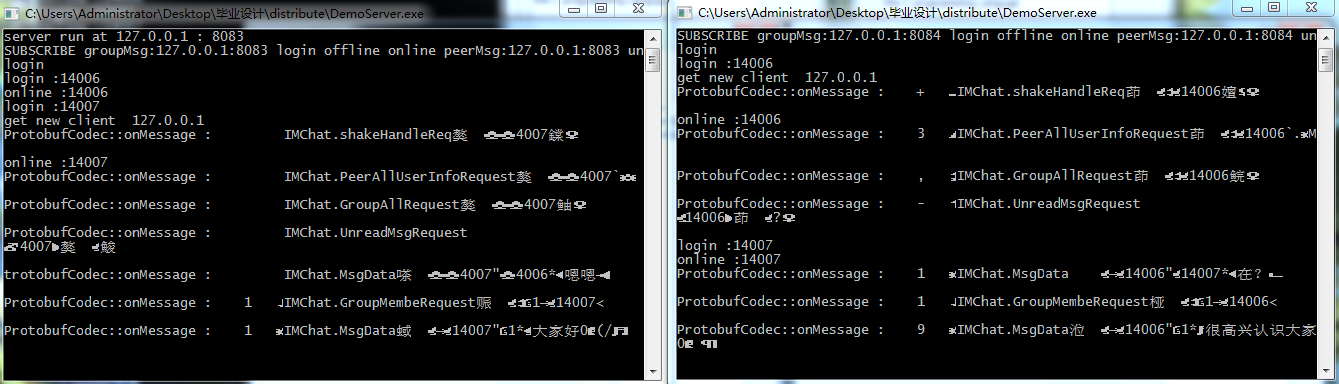


图5.3 在不同消息服务器上的聊天记录

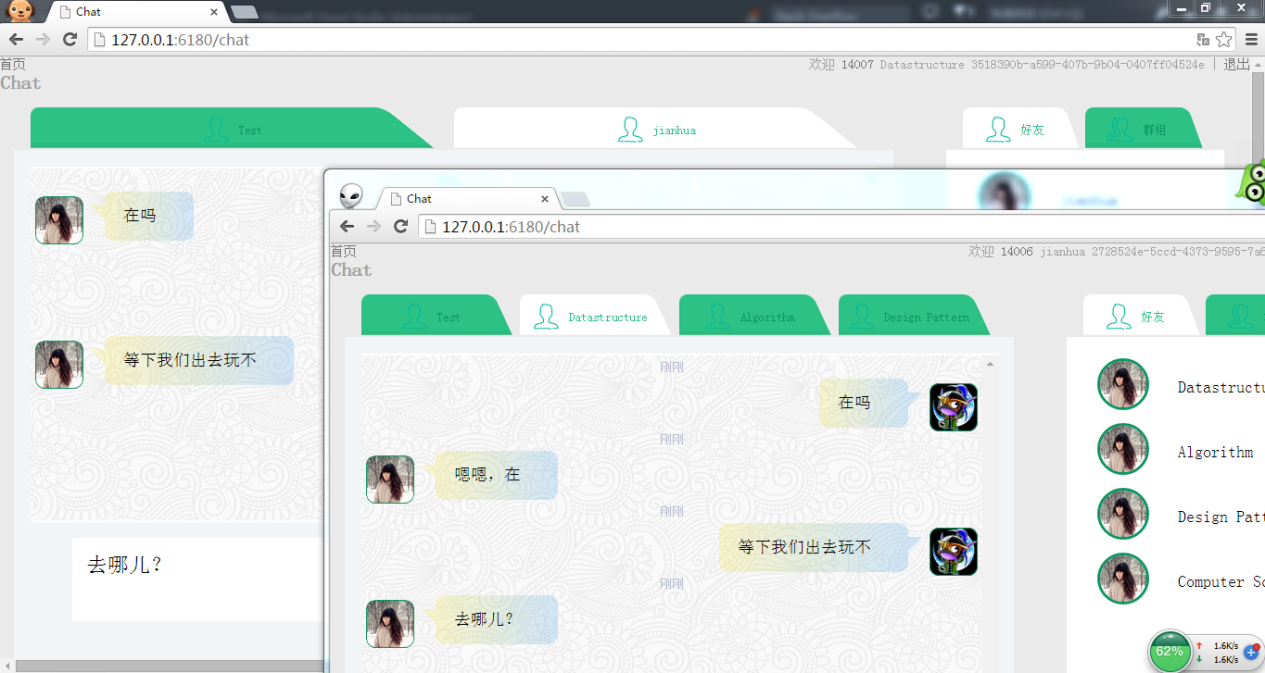


图5.4 不同浏览器之间即时通信效果

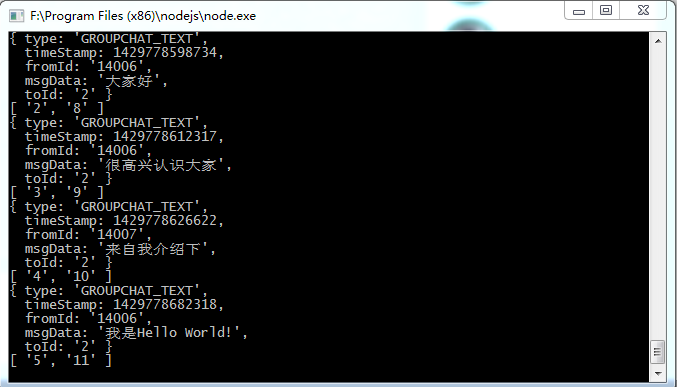


图5.5 Websocket服务器消息记录

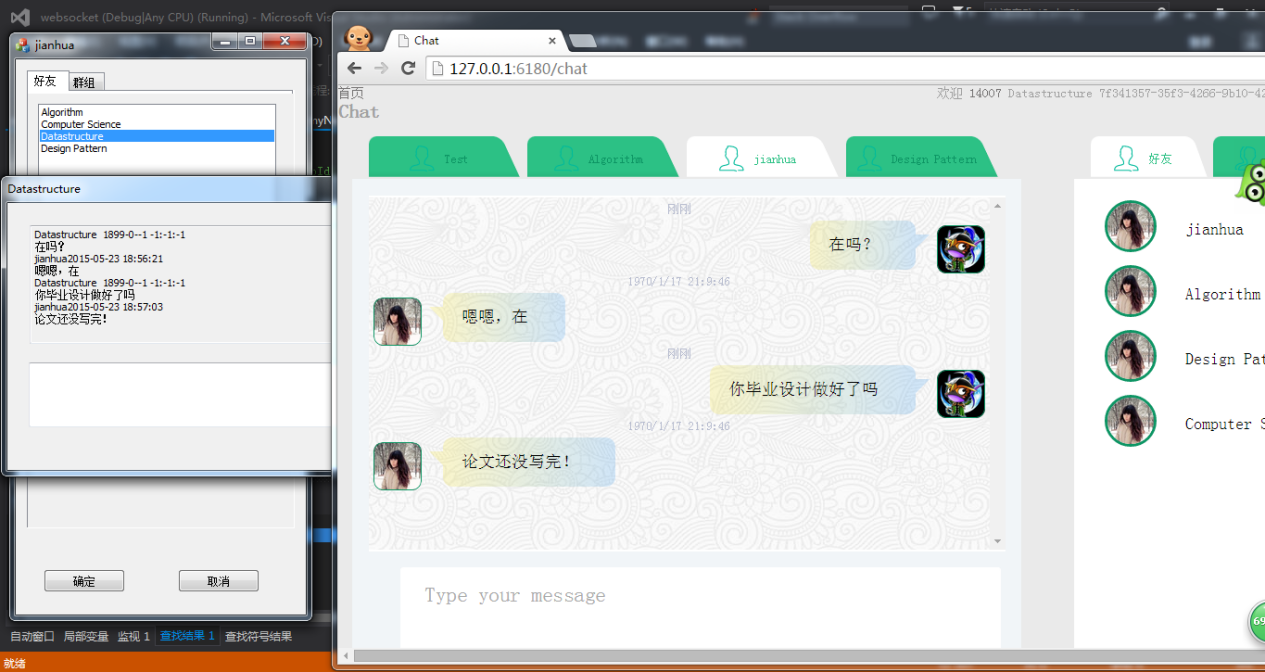


图5.5 客户端与浏览器即时通信效果

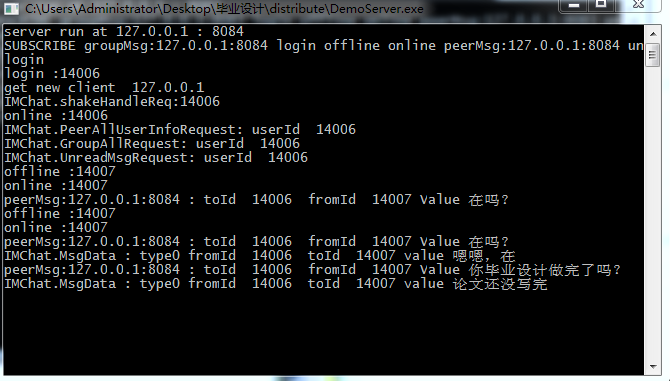


图6.6 客户端与浏览器聊天时C++服务器记录

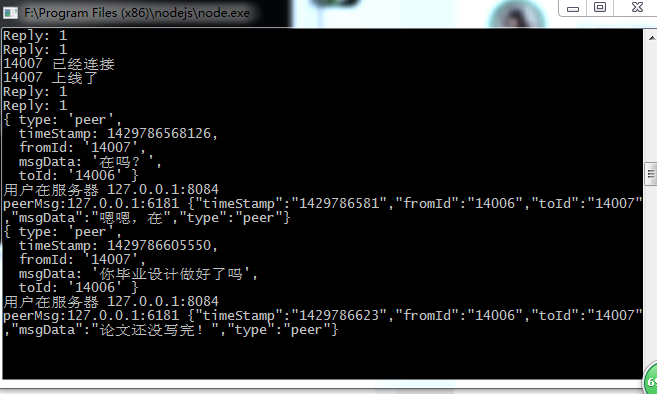


图6.7 客户端与浏览器通信时Websocket服务器记录

# 第六章 结论和展望

## 6.1 结论

本文主要讨论了基于TCP和Websocket协议的即时通信系统的研究和实现。研究不同协议类型服务器通过统一的服务器消息格式实现互通互联，简单的负载均衡，异步网络编程，Publish/Subscribe设计模式，实现了Web端与传统客户端之间即时通信，服务器的高性能、高并发。本系统按照软件工程思想，使用面向对象设计模式设计并封装了大量的类，函数式编程方法解决异步编程的困难，并采用多服务器协作的方式，提高了服务器的健壮性和可拓展性。 在实现的过程中学习了多种网络协议。

## 6.2 进一步工作的方向

由于时间和能力有限，基于TCP和Websocket协议的即时通信系统的还不是很完善，系统的一些其他功能和细节需要进一步的改进和增强。需要继续完善的功能有：

① 增加文件传输、视频聊天、语音聊天、视频会议等功能。

② 将C++登录服务器也改为Https。

③ 编写自己的消息队列服务器。本系统借助redis高速缓存数据库实现服务器间消息队列。

④ 添加系统监控功能。由于时间和目前能力有限没有设计系统监控功能和日志记录，只是简单输出在控制台界面。

⑤ 对数据进行分布式存储。虽然服务器采用多物理机，但设计是没有考虑对数据的分布式切割，这对服务器的拓展性影响很大。

⑥ 更改服务器数据传输类型。有些序列化的消息存储和传输通过二进制传输，虽然性能好，但可读性比较差，可以将传输类型改为JSON数据。

# 参考文献

1. 牛晓蕾. 基于 TCP/IP 协议的安全即时通信系统设计与实现[D]．西安电子科技大学．2007
2. 史济民，顾明华，邓红．软件工程—原理方法与应用[M]．北京：高等教育出版社．2009
3. 谢希仁．计算机网络[M]．北京：电子工业出版社．2007
4. 朴灵．深入浅出Node.js[M]．北京：人民邮电出版社．2013
5. 维基百科 websocket．http://zh.wikipedia.org/wiki/WebSocket．2015-4-14
6. [美]Scott Meyers. Effective C++[M].北京：电子工业出版社. 2006-7
7. [美] Stanley B．Lippman，[美] Josée Lajoie，[美] Barbara E. Moo. C++ Primer 5th Edition[M]．北京：电子工业出版社．2013
8. 百度百科 boost( C++ 库）．http://baike.baidu.com/subview/663725/10275011.htm．2015
9. Google Protocol Buffer 的使用和原理. http://www.ibm.com/developerworks/cn/linux/l-cn-gpb/ .2011
10. openssl. http://www.openssl.org/docs/.2015
11. 百度百科 openssl. http://baike.baidu.com/view/300712.htm. 2015
12. hiredis .https://github.com/redis/hiredis
13. socket.io. https://www.npmjs.com/package/socket.io. 2015
14. 黄健宏． Redis设计与实现[M]．北京：机械工业出版社．2014
15. 百度百科 nginx．http://baike.baidu.com/view/926025.htm．2015
16. 百度百科 sqlite http://baike.baidu.com/view/19310.htm 2015
17. 王珊，萨师煊.数据库系统概论（第四版）[M].北京:高等教育出版社，2007
18. Nginx .http://nginx.org/en/docs/
19. 孙鑫. VC++深入详解[M].北京：电子工业出版社.2006
20. 在 muduo 中实现 protobuf 编解码器与消息分发器. http://www.cnblogs.com/Solstice/archive/2011/04/13/2014362.html#2982572. 2011
21. 一种自动反射消息类型的 Google Protobuf 网络传输方案.http://blog.csdn.net/Solstice/article/details/6300108 . 2011-04-03
22. 封包和拆包. http://blog.csdn.net/fengge8ylf/article/details/793808 .2006-06-03

# 致谢

经过长期的学习，我们的项目终于接近尾声。在这个漫长而又短暂的过程中，我收到了来自各方面的帮助，特在此致谢。

首先要向我的指导老师朱懿敏老师同时也是班主任致以衷心的感谢，在这整个过程中，朱老师以其知识渊博，远见卓识和治学严谨的态度为我树立了榜样。同时，朱老师不管在学习方面还是生活方面，都给予我很多指导和建议，让我更加清楚今后学习的方向和人生的目标。

其次向计算机学院及全体老师致以最大的敬意，感谢他们这些年对我的辛勤教育和指导。

最后，感谢我的搭档及支持和帮助我的任何一个人，你们的鼓励和帮助是我前进的动力。

陈建华

2015年4月15日