中文题目：局域网即时聊天软件

外文题目：LANINSTANT CHAT SOFTWARE

毕业设计（论文）共87页（其中：外文文献及译文18页）图纸共0张

完成日期 2017年6月 答辩日期 2017年6月

摘要

在网络越来越发达的现今社会，人们对于网络的依赖越来越多，也越来越离不开网络，于是产生的聊天工具也越来越多，例如，国外的ICQ，国内的腾讯QQ、微信等。基于C++网络编程的程序设计，本次毕业设计使用C++的网络编程模块编写一个局域网即时聊天软件。

一般来说，聊天工具大多数都是由客户端和服务器以及服务器中存储数据的数据库组成。本软件采用C/S模式，利用Socket套接字实现客户端之间的相互通信。系统设计采用的是面向对象技术和面向对象的设计原理，采用的C/S结构，是利用服务器充当客户端之间的消息中转站，客户端之间的联系必须通过服务器。

该局域网即时聊天软件完成后，可进行局域网内各个用户登陆或注册登陆后，用户之间的聊天、文件传输等。

关键字：C/S模式；Socket；局域网即时聊天软件；服务器；客户端

**ABSTRACT**

Increasingly developed in the network today, people rely on the network more and more, more and more cannot do without the network, more and more and chat tools, for example, the domestic and foreign ICQ, Tencent QQ, WeChat and so on. The program design of C++ network programming based on the graduation design using network programming module C++ the preparation of a LAN chat software.  
 Generally speaking, most chat tools are composed of data stored in the client and the server and the database in the server. The software uses the C/S mode to achieve mutual communication between the client using Socket sockets. System design is based on object oriented technology and design principle for like, using C/S structure, is the use of server acts as a client the message transfer station, the connection between the client must pass through the server.  
 After the LAN instant chat software is completed, the LAN users can login or register, login, chat among users, file transfer and so on.

**Key words: C/S mode; Socket; LAN instant chat software；Server；Client**

目录

[前言 1](#_Toc484530393)

[1 系统问题定义 2](#_Toc484530394)

[1.1 系统名称 2](#_Toc484530395)

[1.2 研究背景 2](#_Toc484530396)

[1.3 研究思路和方法 3](#_Toc484530397)

[1.4 目的以及意义 3](#_Toc484530398)

[1.5 通信接口 3](#_Toc484530399)

[2 可行性分析 4](#_Toc484530400)

[2.1 可行性分析的目的 4](#_Toc484530401)

[2.2 技术可行性 4](#_Toc484530402)

[2.3 经济可行性 4](#_Toc484530403)

[2.4 操作可行性 4](#_Toc484530404)

[2.5 法律可行性 4](#_Toc484530405)

[2.6 可行性研究结论 5](#_Toc484530406)

[3需求分析 6](#_Toc484530407)

[3.1 系统用例分析 6](#_Toc484530408)

[3.1.1 用例图 6](#_Toc484530409)

[3.1.2 用例规约 8](#_Toc484530410)

[3.2 功能需求 10](#_Toc484530411)

[4 系统分析 11](#_Toc484530412)

[4.1 基本组织结构的分析 11](#_Toc484530413)

[4.2 设计原则分析 11](#_Toc484530414)

[4.3 对象模型分析 12](#_Toc484530415)

[4.4 时序设计分析 16](#_Toc484530416)

[4.4.1 时序图 16](#_Toc484530417)

[4.4.2 活动图 20](#_Toc484530418)

[4.3 技术分析 22](#_Toc484530419)

[4.5.1 C/S 软件架构技术 22](#_Toc484530420)

[4.5.2 传输协议 23](#_Toc484530421)

[4.5.3 C++中的MFC和Socket 24](#_Toc484530422)

[4.5.4开发配置要求 27](#_Toc484530423)

[5 系统设计 28](#_Toc484530424)

[5.1聊天模块的整体流程 28](#_Toc484530425)

[5.2 主要流程图 29](#_Toc484530426)

[5.3 系统架构 30](#_Toc484530427)

[5.4 数据流程图 31](#_Toc484530428)

[5.4.1 服务器 31](#_Toc484530429)

[5.4.2 客户端 32](#_Toc484530430)

[5.5 功能设计 33](#_Toc484530431)

[5.5.1 服务器 33](#_Toc484530432)

[5.5.2 客户端 33](#_Toc484530433)

[5.6 数据库设计 33](#_Toc484530434)

[5.6.1 数据库选择 33](#_Toc484530435)

[5.6.2 数据设计原则 34](#_Toc484530436)

[5.6.3 数据库表设计 34](#_Toc484530437)

[6 系统实现 35](#_Toc484530438)

[6.1 开发工具的选择 35](#_Toc484530439)

[6.2 命名规则 35](#_Toc484530440)

[6.3 具体功能展示 36](#_Toc484530441)

[6.4 程序清单 43](#_Toc484530442)

[7 系统测试 44](#_Toc484530443)

[7.1 测试的目的 44](#_Toc484530444)

[7.2 测试方案 44](#_Toc484530445)

[7.3 测试结果 45](#_Toc484530446)

[8 结论 46](#_Toc484530447)

[致谢 47](#_Toc484530448)

[参考文献 48](#_Toc484530449)

[附录A 译文 49](#_Toc484530450)

[附录B 外文文献 56](#_Toc484530451)

[附录C 代码清单 66](#_Toc484530452)

# 前言

在 21 世纪，当今时代也由工业经济逐渐成为了知识经济、信息经济的时代。信息的需求量的不断增加，促使了计算机通信的日新月异！虽然网络的使用频率与使用的范围都越来越广泛，但是人们使用的网络的用途主要的还是只有两个：

(1) 相互之间的沟通；

(2) 资源共享。

针对这种情况，聊天工具便应运而生了。而天工具的出现，促进了网络社会的形成，人们的生活也逐渐的从真实世界，开始转移到虚拟世界。而虚拟世界就是指以网络为载体的生活，人们通常不需要知道对方是谁，或者不需要面对面就可以进行交流。网络的虚拟世界从某种程度上促进了社会的各个方面的发展，无论是经济上还是文化上。而作为网络社会的交通工具之一的聊天软件扮演者极为重要的角色，它促进了人们的思想交流，加快了信息的流通。

本文主要目的是要描述建立一个局域网聊天系统的过程，文中重点描述了该聊天系统的客户端的登陆模块，注册模块，界面管理模块，聊天窗口模块等。以及服务器的登陆监控模块，消息中转模块等。系统选用的是C/S架构搭建的，利用C++中的Socket套接字实现各个客户端之间的通信。

# 1 系统问题定义

早在1970年，就出现了一种即时通讯形式的柏拉图系统（PLATO system）。在这之后，许多的通讯软件被开发出来，并应用广泛，与人们的生活息息相关。如国内日常使用的QQ、微信等。而本设计只是对局域网内的通讯进行开发和实现。

局域网作为网络的一个重要组成部分之一，它不但是用户与外界连接的枢纽，同时也是近距离用户之间沟通与资源共享的一个重要途径。在这种情况下，如何使局域网内的用户便捷的勾通，以及快速的资源共享，也就成为了人们对网络研究的重点之一。

本系统是一个局域网内的即时聊天系统，主要是利用MFC中封装的CSocket完成的，系统

## 1.1 系统名称

局域网即时聊天软件。

## 1.2 研究背景

随着计算机和通讯技术的迅速发展，计算机的网络化已经慢慢渗透到各个应用领域，尤其是以 TCP/IP 协议之间的网络互连发展很快，大部分的计算机应用系统的开发由传统的单机方式转向到了网络环境的开发。大部分编程语言都提供了基于Windows Sockets网络编程接口的 Winsocket 控件，它使开发 TCP/IP 通讯变得简单方便。网络之间进行数据通讯时，使用地址来标识网络中的计算机，有了地址以后，数据才会准确地发送到地址所指向的计算机。TCP/IP 协议使用IP地址为网络中计算机的标识，而且每台计算机的 IP 地址是唯一的，一般情况下，每台计算机运行的程序不止一个，因此要在应用程序中建立连接的话，还需要一个地址标识。在 TCP/IP 协议书中使用端口(PORT)作为计算机上运行的应用程序的标识，因此 TCP/IP 协议中一个有效的网络地址包括IP地址和端口地址两个部分。TCP/IP 协议提供了两种计算机之间的通讯方式：TCP 方式和 UDP 方式。

TCP 是一种面向连接的服务，在文件传输时会先行建立连接，通过链接提供双向、有序且无重复的数据流服务、以及流量控制、差错检测和纠错等服务，从而确保文件传输的正确性；而UDP 则是一种无连接的服务，它在传输文件时不会确保传输端的文件能够正确无误的传送到接收端，所以当使用 UDP 时，通常要自己作接收确认的工作[1]。

## 1.3 研究思路和方法

本系统的开发采用基于Socket的消息发送模式，进行文字聊天和文件传输的设计与开发。因此，本设计首先分析该系统的相关功能，结合本设计的相关要求写出需求分析；其次，综合运用以前所学的相关知识，选择所熟悉的开发工具进行本设计的开发；在设计中以需求分析为基础，写出系统开发计划、实现流程及相关问题的实现方法；最后，在系统开发完毕后，进行调试和运行，做好调试和运行的相关记录。

## 1.4 目的以及意义

随着互联网技术的迅猛发展，网络给人们带来了很多便利，比如人们借助于网络进行相互交流变得更加方便。聊天工具作为互联网中运用最为广泛的通信工

具之一，如今都已被广大网民接受。很多人都使用过网络聊天软件，网络聊天软件发展到现在已经具有了很多功能：有文字聊天、文件、文件夹传输的基本功能，也有密封、加锁等较为复杂的功能。本毕业设计的目的主要是为了检查使学生综合运用以前所学知识（包括以前所学的一些关于网络、编程、信息安全等知识）的能力，模拟局域网飞鸽聊天软件，开发一个聊天系统软件，本毕业设计主要实现局域网内文字聊天及文件传输等功能。

## 1.5 通信接口

Winsock 兼容多个协议，所以使用通用的寻址方式。TCP/IP 使用地址和端口号来指定一个地址，但是其它协议也许采用不同的形式。如果 Winsock 强迫使用特定的寻址方式，添加其他协议就不大可能了。在 Winsocket 中应用程序员通过 SOCKADDR\_IN 结构来指定 IP 地址和端口号。

应用程序可以使用 inet\_addr 函数将一个小数点分隔的十进制 IP 地址字符 串转化成由 32 位二进制数表示的 IP 地址。inet\_ntoa 函数将一个网络字节顺序的 32 位 IP 地址转化字符串。注意：inet\_addr 返回的 32 位二进制数是用网络顺序存储的[2]。

# 2 可行性分析

## 2.1 可行性分析的目的

可行性分析是指在当前的现实环境和现有条件下，所开发的整个项目是否具备必要的资源以及其他条件。它是保证整个项目开发工作能够正常运行的前提。通过调研发现，局域网聊天系统有充分的可能性和可行性。其可行性分析可由技术可行性、经济可行性、操作可行性、法律可行性等方面进行分析。

## 2.2 技术可行性

技术可行性主要是分析当前的软硬件技术能否满足对开发本系统所提出的技术要求，以及开发人员是否已掌握开发这个系统的技术。

本系统以C++为主要编程语言，再结合网络通信基本原理和Socket编程及MFC的API技术，技术成熟，可行。

## 2.3 经济可行性

这个系统开发开发成较低，长远效益会远大于开发成本，可行。

## 2.4 操作可行性

系统硬件为PC机，系统win 7及以上，而且编程环境Visual Studio 2015为专业程序员使用工具，比较易于使用，可行。

## 2.5 法律可行性

系统为自主开发的，是个小型的局域网即时聊天系统，只作为学习和测试用途，不用与商业，不存在侵权的问题。

## 2.6 可行性研究结论

综上所述，这个系统无论从技术、经济、操作、法律方面都是可行的，所以开发此系统可行。

# 3需求分析

系统需求是系统必须遵守的一系列约束条件，以及要实现的一系列功能。在需求分析阶段，最困难的不是程序的构造，而是发现程序的真正需求。统一的过程提供了一些最佳的实践，其中之一就是管理需求（manage requirement），这不同于传统的瀑布模型一下子将需求定义完全，而是采用迭代的思想逐渐去实现系统需求模型，用例（Use Case）研究就是一种发现需求的方法。

## 3.1 系统用例分析

Jacobson在他1992年的著作中写道：“用例是与系统进行对话时行为相关的事务系列的描述。”在统一过程中，对用例的描述没有实质性的改变，它认为用例是“一系列带变量的动作描述，系统由此对特定用户产生有价值的可见结果”。

用例就是系统需求，在编写用例的时候，其实就是对系统的功能需求进行分析，用来表示系统需要实现的功能，具体由系统的实现阶段完成。用例图中主要有参与者和用例两部分。参与者（Actor）是直接与系统互相作用的系统、子系统或类的外部实体的抽象概念。用例是动作顺序的规格说明，系统、子系统或类能够通过与外部参与者的交互而执行。

### 3.1.1 用例图

用例图是UML中的一种对用例建模的图，用来简要描述系统功能[3]。

表2-1用例图符号说明

Tab.2-1 Use case diagram symbols

|  |  |  |
| --- | --- | --- |
| 图例 | 说明 | 含义 |
| 角色 | 角色（Actor） | 表示Actor，是用户作用于系统的一个角色（Role）。Actor有自己的目标。通过与系统的交互达到目标。 |
| 1 | 用例（Use Case） | 表示Use Case，是对系统的用户需求（主要是功能需求的描述），Use Case表达了系统的功能和提供的服务。 |
| 2 | 关联关系 | 表示Unidirectional Association，是Actor与Use Case之间的关联关系。关系方向显示系统或角色启动通信。 |

1. 服务器：



图3-1 服务器用例图

Fig.3-1 Server use case diagram

1. 客户端：



图3-2 注册用例图

Fig.3-2 Registered use case diagram



图3-3 客户端用例图

Fig.3-3 Client use case diagram

### 3.1.2 用例规约

（1）服务器

* 1. 开启服务器

描述：管理员打开服务器并开启服务

主要参与者：管理员

先决条件：管理员拥有服务器的可执行文件

后置条件：开启服务器

基本事件流：

1. 管理员打开服务器端
2. 管理员开启服务

（2）客户端

2.1 注册用户

描述：用户登陆之前先要注册账号

主要参与者：用户

先决条件：服务器打开，用户拥有客户端可执行程序

后置条件：服务器注册验证成功

基本事件流：

1. 打开客户端，进入登陆界面，用户点击注册按钮
2. 输入用户昵称
3. 输入登陆密码
4. 生成唯一标识ID，用于登陆使用
5. 注册成功
6. 登陆

2.2 用户登陆

描述：用户使用ID和密码进行登陆

先决条件：服务器打开，用户拥有客户端可执行程序，并且用户已注册

后置条件：服务器验证成功

基本事件流：

1. 打开客户端，输入ID和密码，点击登陆
2. 进入客户端主界面
3. 用户可以双击在线用户进入聊天界面进行聊天、发送文件、修改聊天字体，也可以查询聊天记录等
4. 在客户端界面，用户可以修改头像，可以打开宠物界面观看宠物的动画
5. 结束聊天时退出

## 3.2 功能需求

根据软件的基本需求，本系统分为两个子系统:

1. 服务器：是整个软件的根本，它是在局域网内用户登陆消息确认和各种消息转发的根本，整个软件围绕服务器构建起来的。
2. 客户端：它是提供用户使用的界面程序，也是整个软件展现在用户面前的窗口，有了客户端，用户不需要知道服务器端是什么原理，只需要根据开发者为用户提供的使用手册即可进行使用。

本系统的聊天系统实现的功能类似于飞鸽用户之间的聊天。在设计系统时，应根据简洁、流畅的特点安排布局，在编制程序时应充分考虑这一点传输的稳定、快捷，真正做到"简洁、高效、流畅、安全"的使用环境。用户使用的过程为，先是用户登开启本软件，软件自动检索局域网的其它用户，并以之成为用户列表，让用户选择。当用户发现所要连接的用户之后，双击该用户，则出现发送窗口。用户用之来发送消息或者文件。接收方可以选择接收或者拒绝相应的传输文件。

# 4 系统分析

## 4.1 基本组织结构的分析

以下为基本组织结构图：

到达用户

客户端

选择交互

数据传输

服务器

文字聊天

文件传输

图4-1 组织结构

Fig.4-1 Organizational structure

整个系统都是围绕这两个核心功能进行开发，做为一个聊天软件，主要的功能就是实现用户与用户之间的交互，而最基本的就是文本信息的交互和文件的共享等功能。其它功能就不一一列出，在系统设计中将会详细阐述整个系统的所有应当实现的功能。

## 4.2 设计原则分析

根据对网络聊天软件系统的调研，本设计的实现中需要满足以下几个系统设计原则[4]：

1. 实用性原则：真正为局域网的实际工作服务，按照需求的轻重缓急，合理设计本系统。
2. 可靠性原则：必须为用户提供安全的服务，尤其是要保证传输的稳定性。
3. 友好性原则：使用本系统的用户相当一部分对计算机知识了解很少，所以系统操作上要求简单方便，便于用户掌握。
4. 可扩展性原则：采用开放的标准和接口，便于系统向更大的规模和功能扩展。

## 4.3 对象模型分析

系统用例模型

类图是描述类、接口、协作以及它们之间关系的图，用来显示系统中各个类的静态结构。类不仅显示了系统内信息的结构，也描述了系统内信息的行为。

表4-1类图符号说明

Tab.4-1 Class diagram symbols

|  |  |  |
| --- | --- | --- |
| 图例 | 说明 | 含义 |
| 6 | 类 | 表示类（Class）,类是面向对象系统组织结构的核心，顶部区域显示类的名字，中间的区域列出类的属性，底部的区域列出类的操作。 |
|  | 关联关系 | 表示关联关系（Association），它指明一个事物对象和另一个事物的对象之间的联系。 |

整个聊天系统的服务器和客户端的所有类图如下：

类之间的关系有关联关系、泛化关系、依赖关系和聚合关系。泛化关系也称为一般化关系，表示类与类之间的继承关系，接口与接口之间的继承关系，或类对接口的实现关系。一般化是继承关系，在UML中可以在包之间建立一般化关系。关联关系是一种结构化的关系，指一种对象和另一种对象有联系。给定有关联的两个类，可以从一个类的对象得到另一个类的对象。关联有两元关系和多元关系。两元关系是指一种一对一的关系，多元关系是一对多或多对一的关系。一般用实线连接有关联的同一个类或不同的两个类。依赖关系是一种使用关系，特定事物的改变有可能会影响到使用该事物的事物，反之不成立。通常情况下，依赖关系体现在某个类的方法使用另一个类作为参数。在UML中可以在其它的事物之间使用依赖关系，特别是包和节点之间。聚合是关联关系的一种，是较强的关联关系，强调的是整体与部分之间的关系。与关联关系一样，聚合关系也是通过实例变量来实现这样关系的。

1. 服务器



图4-2 服务器类图

Fig.4-2 Server class diagram



图4-3 服务器类关系图

Fig.4-3 Server class relation diagram

1. 客户端



图4-4 客户端类图

Fig.4-4 Client class diagram

****

图4-5 客户端类关系图

Fig.4-5 Client class relation diagram

## 4.4 时序设计分析

需求阶段主要对系统的功能建立用例模型，每个用例是系统提供给用户使用，并且是提供给用户的一个可视化操作界面，具体的实现过程，可以通过时序图来表示，时序图表示出了整个系统的处理过程，清楚地显示系统中的各个功能所涉及的对象及对象之间的消息传递的过程。针对需求分析中的用例图，每一个用例图用一个时序图表示该系统的流程。

### 4.4.1 时序图

时序图（Sequence Diagram）描述了对象之间传送消息的时间顺序，它用来表示用列图中的行为顺序。

时序图将交互关系表示为二维图。当对象存在时，生命线用一条虚线表示，当对象的过程处于激活状态时，生命线是一个双道线。消息用从一个对象的生命线到另一个对象生命线的箭头表示。箭头事件顺序在图中从上到下排列。

表4-2时序图符号说明

Tab.4-2 Sequence diagram symbols

|  |  |  |
| --- | --- | --- |
| 符号 | 说明 | 含义 |
|  | 对象 | 表示参与交互的对象，时序图中的对象使用矩形将对象名称包含起来，并且对象名称下有下划线。 |
| 17 | 生命线 | 表示生命线（Lifeline）,它是一条垂直的虚线，表示时序图中的对象在一段时间内的存在。 |
| QQ截图20120613195820 | 消息 | 表示Object Message,含义是在两个对象之间绘制消息。 |
| QQ截图20120613195927 | 返回的消息 | 表示Return Message，显示过程调用返回的消息。 |
| QQ截图20120613200149 | 反身消息 | 表示反身消息（Message to Self）。 |
| QQ截图20120613200050 | 激活期 | 激活期（Activation），激活表示该对象被占用以完成某个任务，去激活指的是对象处于空闲状态，在等待消息。 |

1. 服务器

图4-6 服务器时序图

Fig.4-6 Server sequence diagram

1. 客户端



图4-7 注册账号时序图

Fig.4-7 Registered account sequence diagram



图4-8 客户端时序图

Fig.4-8 Client sequence diagram

### 4.4.2 活动图

活动图（Activity Diagram）是UML用于对系统的动态行为建模的另一种常用工具，它描述活动的顺序，展现从一个活动到另一个活动的控制流。活动图在本质上是一种流程图。

活动图中包含的图形元素有动作状态、活动状态、动作流、分支与合并、交叉与汇合、泳道和对象流等。

表4-3活动图符号说明

Tab.4-3 Activity diagram symbols

|  |  |  |
| --- | --- | --- |
| 图例 | 说明 | 含义 |
| 7 | 活动开始 | 表示活动开始（Start State），是活动图的起始位置。活动起始在一个状态图中只能有一个。 |
| 8 | 活动终止 | 表示活动终止(End State)，是活动图的终点。活动终止在一个状态图中可以有多个。 |
| 9 | 活动 | 表示活动（Activity），其作用是添加活动。 |
| 2 | 状态转换 | 表示状态转换（State Trastion） |
| 12 | 水平同步 | 表示分叉与汇合。 |
| 111 | 分支与合并 | 表示分支与合并，表示对象类所具有的条件行为。 |

1. 服务器



图4-9 服务器活动图

Fig.4-9 Server activity diagram

1. 客户端



图4-10 客户端活动图

Fig.4-10 Client activity diagram

## 4.3 技术分析

本系统采用 C++ 语言开发。C++是迅速发展和推广的一种计算机语言，近年来，已经相当的完善。它主要特点有两个方面：一是全面兼容 C，二是支持面向对向的方法。

### 4.5.1 C/S 软件架构技术

C/S（Client/Server）结构，即大家熟知的客户机/服务器结构。它是软件系统体系结构，通过它可以充分利用两端硬件环境的优势，将任务合理分配到Client 端和 Server 端来实现，降低了系统的通讯开销。目前大多数应用软件系统都是 Client/Server 形式的两层结构，由于现在的软件应用系统正在向分布式的 Web 应用发展，Web 和 Client/Server 应用都可以进行同样的业务处理，应用不同的模块共享逻辑组件；因此，内部的和外部的用户都可以访问新的和现有的应用系统，通过现有应用系统中的逻辑可以扩展出新的应用系统。这也就是目前应用系统的发展方向。

传统的C/S体系结构虽然采用的是开放模式，但这只是系统开发一级的开放性，在特定的应用中无论是 Client 端还是 Server 端都还需要特定的软件支持。由于没能提供用户真正期望的开放环境，C/S 结构的软件需要针对不同的操作系 统系统开发不同版本的软件，加之产品的更新换代十分快，已经很难适应百台电脑以上局域网用户同时使用。 但本文旨在通过C/S结构分析聊天软件的全局，故而不考虑其他因素。

### 4.5.2 传输协议

TCP（Transmission Control Protocol）：

是一种面向连接的、可靠的、基于字节流的传输层通信协议，由IETF的RFC 793定义。在简化的计算机网络OSI模型中，它完成第四层传输层所指定的功能，用户数据报协议(UDP)是同一层内另一个重要的传输协议。在因特网协议族(Internet protocol suite)中，TCP层是位于IP层之上，应用层之下的中间层。不同主机的应用层之间经常需要可靠的、像管道一样的连接，但是IP层不提供这样的流机制，而是提供不可靠的包交换。

应用层向TCP层发送用于网间传输的、用8位字节表示的数据流，然后TCP把数据流分区成适当长度的报文段(通常受该计算机连接的网络的数据链路层的最大传输单元(MTU)的限制)。之后TCP把结果包传给IP层，由它来通过网络将包传送给接收端实体的TCP层。TCP为了保证不发生丢包，就给每个包一个序号，同时序号也保证了传送到接收端实体的包的按序接收。然后接收端实体对已成功收到的包发回一个相应的确认(ACK);如果发送端实体在合理的往返时延(RTT)内未收到确认，那么对应的数据包就被假设为已丢失将会被进行重传。TCP用一个校验和函数来检验数据是否有错误;在发送和接收时都要计算校验和。

UDP（User Datagram Protocol）：

用户数据包协议，它和 TCP 一样位于传输层，和 IP 协议配合使用，在传输数据时省去包头，但它不能提供数据包的重传，所以适合传输较短的文件。

由于大多数网络应用程序都在同一台机器上运行，计算机上必须能够确保目的地机器上的软件程序能从源地址机器处获得数据包，以及源计算机能收到正确的回复。这是通过使用 UDP 的"端口号"完成的。源端口号标识了请求域名服务的本地机的应用程序，同时需要将所有由目的站生成的响应包都指定到源主机的这个端口上。与 TCP 不同，UDP 并不提供对 IP 协议的可靠机制、流控制以及错误恢复功能等。由于 UDP 比较简单，UDP 头包含很少的字节，比 TCP 负载消耗少，所以 UDP 适用于不需要 TCP 可靠机制的情形，比如，当高层协议或应用程序提供错误和流控制功能的时候。UDP 是传输层协议，服务于很多知名应用层协议，包括网络文件系统（NFS）、简单网络管理协议（SNMP）、域名系统（DNS）以及简单文件传输系统（TFTP）。

UDP广播（broadcasting）是多点投递的最普遍的形式，它向每一个目的站投递一个分组的拷贝。它可以通过多个单次分组的投递完成，也可以通过单独的连接传递分组的拷贝，直到每个接收方均收到一个拷贝为止。在多数网络中，用户是通过把分组分送给一个特殊保留的地址即广播地址（broadcast address）来进行广播投递，它的主要缺点是会耗费大量的主机资源和网络资源[5]。

### 4.5.3 C++中的MFC和Socket

1. Socket及Window Socket API简介

套接字（Socket）是一种双向的通信接口，可以通过这个端口与任何一个具有Socket端口的计算机通信，套接字是网络通信的基础。Socket在Windows以句柄的形式被创建。使用Socket进行网络通信必须包含下面的几种信息：双方认可的协议，本地主机的IP地址，本地进程的协议端口，对方主机的IP地址，对方进程的协议端口[6]。

1. Socket套接字的分类：
2. 数据报套接字（Datagram Sockets）—— 对于在TCP/IP上实现的WinSock，数据报套接字使用用户数据报协议（UDP）。数据报套接字提供了一种不可靠的、非连接的数据包通信方式。
3. 流式套接字（Stream Sockets）—— 流式套接字使用传输控制协议（TCP）。流式套接字可以将数据按顺序无重复地发送到目的地，它提供的是一种可靠的、面向连接的数据传输方式。不管是对单个的数据报，还是对数据包，流式套接字都提供了一种流式数据传输。
4. MFC Socket程序编程的步骤：
5. 加载套接字库，创建套接字(AsfSocket()/socket())；
6. 绑定套接字到一个IP地址和一个端口上(bind())；
7. 将套接字设置为监听模式等待连接请求(listen())；
8. 请求到来后，接受连接请求，返回一个新的对应于此次连接的套接字(accept())；
9. 用返回的套接字和客户端进行通信(send()/recv())；
10. 返回，等待另一连接请求；
11. 关闭套接字，关闭加载的套接字库(closesocket())。
12. socket函数

socket的创建函数，其定义为：SOCKET PASCAL FAR socket (int af, int type, int protocol)

1. 参数int af，代表网络地址族，目前只有一种取值是有效的，即AF\_INET，代表internet地址族；
2. 参数int type，代表网络协议类型，SOCK\_DGRAM代表UDP协议， SOCK\_STREAM代表TCP协议；
3. 参数int protocol，指定网络地址族的特殊协议，目前无用，赋值0即可。
4. 返回值为SOCKET，若返回INVALID\_SOCKET则失败。
5. sockaddr\_in、in\_addr类型，inet\_addr、inet\_ntoa函数
6. sockaddr\_in定义了socket发送和接收数据包的地址。
7. sockaddr\_in的含义比in\_addr的含义要广泛，其各个字段的含义和取值如下：
8. 字段short sin\_family，代表网络地址族，如前所述，只能取值AF\_INET；
9. 字段u\_short sin\_port，代表IP地址端口，由程序员指定；
10. 字段struct in\_addr sin\_addr，代表IP地址；
11. 字段char    sin\_zero[8]，很搞笑，是为了保证sockaddr\_in与SOCKADDR类型的长度相等而填充进来的字段。
12. 常用的WinSock函数

C++对网络编程的支持有socket支持，Winlnet支持，MAPI和ISAPI支持等。其中Windows Sockets API是TCP/IP网络环境里，也是Internet上进行开发最为通用的API。最早美国加州大学Berkeley分校在UNIX下为TCP/IP协议开发了一个API，这个API就是著名的套接字接口（Berkeley Socket）。在桌面操作系统进入Windows时代后，仍然继承了Socket方法。从表一可以看出，主要的WinSock API函数有哪些。

表4-4 WinSock API函数

Tab.4-4 WinSock API function

|  |  |
| --- | --- |
| 函数 | 功能 |
| WSAStartup() | 连结应用程序与 Windows Sockets DLL 的第一个函数 |
| WSACleanup() | 结束 Windows Sockets DLL 的使用 |
| socket() | 建立Socket |
| closesocket() | 关闭某一Socket |
| bind() | 将一本地地址与一个SOCKET描述字连接在一起 |
| listen() | 设定 Socket 为监听状态，准备被连接 |
| accept() | 接受某一Socket的连接要求，以完成面向连接的客户端 Socket 的连接请求。 |
| connect() | 要求连接某一Socket到指定的网络上服务端 |
| recv() | 从面向连接的 Socket 接收信息 |
| send() | 使用面向连接的 Socket 发送信息 |
| WSAAsyncSelect() | 要求某一 Socket 有事件 (event) 发生时通知使用者 |

1. MFC Socket编程技术

MFC是对API的封装，MFC针对WinSock API封装了两个类：CAsyncSocket和Csocket。因此，使用这两个类进行Socket网络程序的编写会比使用API方便一些。

MFC中Socket相关类的继承关系如下图：

CObject

CAsyncSock

CSocket

图4-11 MFC中Socket相关类的继承关系

Fig.4-11 Inheritance relationships of Socket related classes in MFC

CAsyncSocket对WinSock API进行了较低程度的封装，缺省情况下建立的Socket是异步操作方式，即非阻塞的，因此一些阻塞操作，需要自己动手完成。另外，由于它对API的封装程度不高，以前在WinSock API中需要自己完成的工作（比如：字节顺序的转换），仍然需要自己完成。当然有许多程序员喜欢使用该类甚至WinSock API，原因是这些方法具有很大的灵活性，给程序员以较大的发挥空间。为了给程序员提供更方便的接口以自动处理网络通信中的一些任务，MFC在CAsycnSocket类的基础上派生了CSocket类，它提供了比CAsyncSocket更高层的WinSock API接口。CSocket的网络操作是同步方式，即阻塞式的。CSocket建立的Socket不是阻塞的，而是对非阻塞的Socket进行处理后实现的阻塞操作。在阻塞任务执行过程中，Windows线程的消息循环仍然有效，也就是说CSocket类的阻塞操作不影响Windows的消息循环。

### 4.5.4开发配置要求

开发此系统所需的基本软、硬件环境为：

(1).win10或win7；

(2).Pentium100 及以上档次的 PC 及其兼容机；

(3).1G 以上内存措施；

(4).1G 以上可用硬盘空间；

(5).VGA 显示器。

以上的配置均要求为在同一局域网内的两台或两台以上。

# 5 系统设计

　 本局域网即时聊天软件的开发采用基于socket 的可靠连接的发送模式，进行文字聊天和文件传输的设计与开发。因此，本次设计的思想有以下几个方面：

（1）分析该系统的相关功能，本次设计着重于文字聊天和文件传输功能的实现。在局域网中在线用户之间的信息交流。对于聊天功能的实现可以使基于点到点的，也就是说，在两个用户之间的文字传输，可以把每一在线用户设成一个类，实际是对这个类的操作，在具体实现中可以在一个对话框中具有文字信息的接收与传输。对于文件传输功能的实现，是在与目标用户建立连接后把数据转换成socket流进行传输。

（2） 综合运用以前所学的相关知识，选择所熟悉的开发工具进行本设计的开发；在设计中以需求分析为基础，在具体实现中有以下几个重要问题,需要解决：

首先，发现机制的实现。如何使局域网的在线用户刷新在显示本地在线用户列表上。

其次，文本消息的发送与接收。如何利用socket实现文本信息的发送与接收。

然后，文件的发送与接收。

（3）在系统开发完毕后，进行调试和运行，做好调试和运行的相关记录。

## 5.1聊天模块的整体流程

　　本系统文字聊天是采用 C/S 结构，所以每个客户端都是与服务器相连接的，当用户发送信息给局域网的用户，客户端的消息先会发送到服务器端，到服务器端接收完成后，根据用户发来的消息包，得到用户消息的目标用户，服务器将自动发送的目标用户，而目标用户接收到消息时，自动弹出对话窗口，而消息也将加载到对话窗口。

由此可以看出本模块主要分为三模块：发送模块、转发模块和接收模块。发送模块：用户在本地客户端选取目标用户，打开对话窗口，发送消息；转发模块：服务器接收到用户的消息之后，自动将消息转发到目标用户；接收模块：目标用户客户端在接收到消息是，将消息加载到对话窗口，若没有打开对话窗口，客户端自行打开。

综上所述，系统的流程图为：

其他用户

服

务

器

用

户

图5-1 聊天流程图

Fig.5-1 Chat technological process diagram

## 5.2 主要流程图

由于本系统采用 C/S 架构故将系统运行载体分为两个部分：客户端与服务器端。文件传输需要建立有效的连接后，通过 TCP/IP 协议实现点对点的文件传输。通信双方是通过三次握手建立连接，实现文字、文件传输，最后关闭套节字连接。

其传输的整个流程图如下：

客户端主机

套字节连接请求

写入信息流

断开连接

服务器端主机

连接相应（3次握手）

接收信息流

断开连接请求

图5-2 业务流程图

Fig.5-2 Business process diagram

## 5.3 系统架构

客户端

用户

客户机

用户

用户

客户机

客户机

Socket

文件服务

服务器服务

服务器

服务器

文件

图5-3 系统架构图

Fig.5-3 System architecture diagram

## 5.4 数据流程图

根据调查和研究，得到系统所要处理的数据流程图。

### 5.4.1 服务器

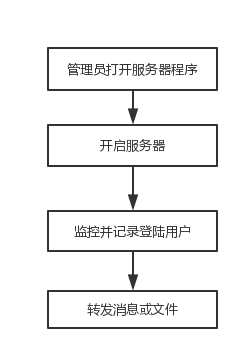


图5-4 服务器流程图

Fig.5-4 Server technological process diagram

### 5.4.2 客户端

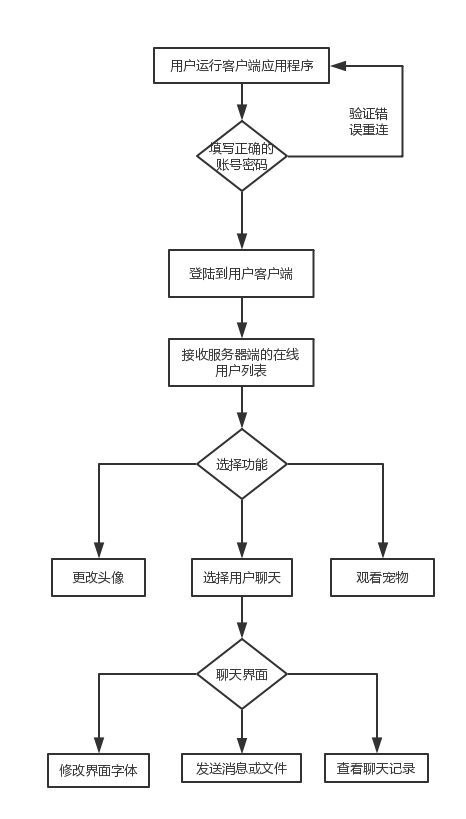


图5-5 客户端流程图

Fig.5-5 Client technological process diagram

## 5.5 功能设计

### 5.5.1 服务器

（1）账号管理功能：当用户登陆的时候，验证是否存在；当注册的时候，保存该账号。

（2）在线用户检测功能：当用户登陆成功后，将该用户添加到在线列表，并发送到各个客户端。

（3）消息中转功能：当接收到消息时，能将消息自动转发到目标用户。

### 5.5.2 客户端

（1）用户登陆功能：用户使用账号密码进行登陆。

（2）注册功能：新用户可以注册，然后使用该系统。

（3）在线用户列表：将服务器发送过来的用户列表显示在客户端界面。

（4）用户头像更换功能：用户可以更换现有的各种头像。

（5）宠物功能：可以点击宠物，跳入宠物表演界面。

（6）聊天功能：用户可以和在线用户进行聊天。

（7）文件传输功能：用户可以对在线用户发送文件。

（8）聊天界面字体更换功能：用户可以选择自己喜欢的字体等进行修改。

（9）聊天记录功能：对于已保存的聊天记录可以进行查阅，聊天记录默认不保存。

（10）保存聊天记录功能：用户可以将聊天记录保存在本地，以便查看。

（11）最小化功能：当用户暂时不需要使用的时候，可以将该系统最小化到托盘，只需双击即可恢复使用。

## 5.6 数据库设计

### 5.6.1 数据库选择

该系统为轻量级学习性设计，所以在考虑成本和效率的条件下，没有链接外部的大型数据库，只是使用了文本文件作为数据的存储。

### 数据设计原则

（1）规范化：使用正确的数据结构，确保数据正确地分布到数据库的表中。

（2）据的独立性：在添加、修改、删除数据时，数据的结构、相互关系和属性不变，并且使应用程序不依赖于数据库中数据的组织方式和存储位置。

（3）据的完整性和安全性：防止数据库中存在不符合语义规定的数据和防止因错误信息的输入输出造成无效操作或错误信息。

### 5.6.3 数据库表设计

（1）服务器端

表5-1 用户信息表

Tab.5-1 User information table

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 字段名 | 类型 | 是否为空 | 是否主键 | 字段描述 |
| ID | int | 否 | 是 | 用户ID |
| UserName | varchar2(10) | 否 | 否 | 用户名 |
| PassWord | varchar2(10) | 否 | 否 | 密码 |

（2）客户端

表5-2 常登用户表

Tab.5-2 Frequently login user table

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 字段名 | 类型 | 是否为空 | 是否主键 | 字段描述 |
| ID | int | 否 | 是 | 用户ID |

# 6 系统实现

## 6.1 开发工具的选择

整个系统架构都是使用Visual Studio 2015这个微软的IDE，使用的是MFC程序框架进行开发。这个IDE是微软公司推出的一个基于Windows系统平台、可视化的集成开发环境，他的源程序是按C++语言的要求编写，并加入了微软提供的强大的MFC（Microsoft Foundation Class）类库。使用这个工具时，可以添加一个第三方的工具包：番茄助手。

Visual Studio 2015包含了很多强大的新特性，无论是从事WEB开发还是从事桌面程序开发，Visual Studio 2015都可以提高用户的开发效率。使用这个IDE，开发者可以减少许多重复性的工作量，软件自动为开发者编辑好这部分共有的代码。而番茄助手则是可以帮助开发者在编写代码的时候更清晰的分辨不同属性的代码，代码的跳转功能也十分强大。

## 6.2 命名规则

在面向对象编程中，对于类，对象，方法，变量等方面的命名是非常有技巧的。比如，大小写的区分，使用不同字母开头等等。但究其本，追其源，在为一个资源其名称的时候，应该本着描述性以及唯一性这两大特征来命名，才能保证资源之间不冲突，并且每一个都便于记忆。

编程过程中，基本使用的是驼峰法命名规则。骆峰式命名法（Camel-Case）是电脑程式编写时的一套命名规则（惯例）。骆峰式命名法就是当变量名或函式名是由一个或多个单字连结在一起，而构成的唯一识别字时，第一个单词以小写字母开始；第二个单词的首字母大写或每一个单词的首字母都采用大写字母，例如：myFirstName、myLastName，这样的变量名看上去就像骆驼峰一样此起彼伏，故得名。

## 6.3 具体功能展示

本系统是基于C/S模式开发的，用C++编码实现。以下是部分功能界面的截图：

（1）服务器

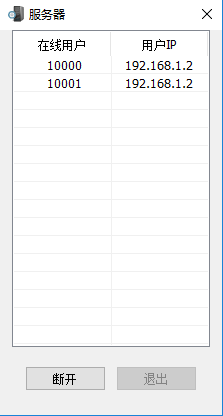


图6-1 服务器监控图

Fig.6-1 Server monitor diagram

（2）客户端



图6-2 客户端用户界面图

Fig.6-2 Client user interface diagram



图6-3 宠物界面图

Fig.6-3 Pet interface diagram

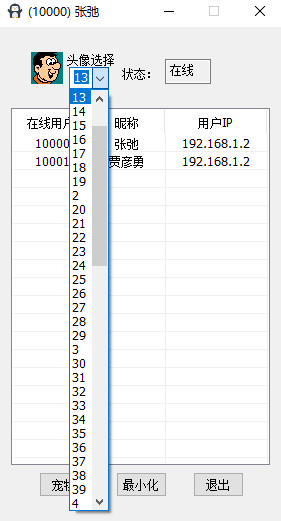


图6-4 更改头像界面图

Fig.6-4 Change face interface diagram

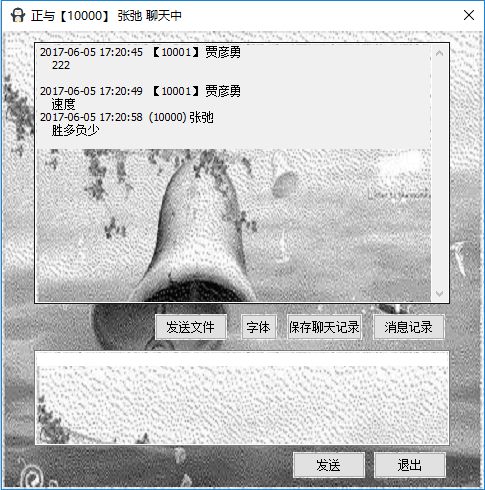


图6-5 聊天界面图

Fig.6-5 Chat interface diagram

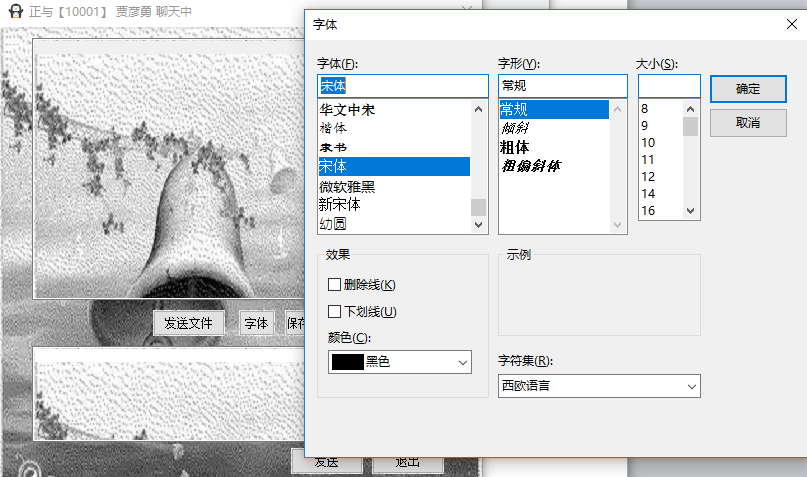


图6-6 字体更改界面图

Fig.6-6 Change progeny interface diagram

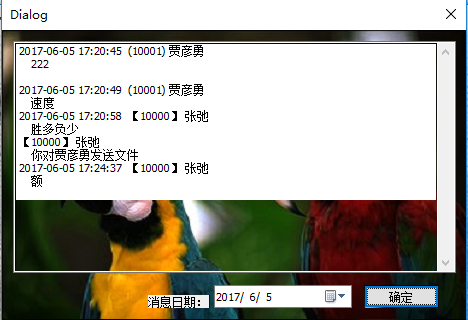


图6-7 消息记录界面图

Fig.6-7 Message recode interface diagram

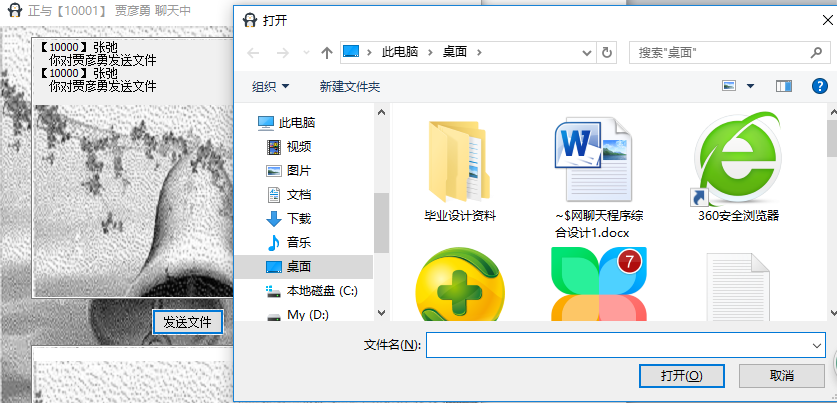


图6-8 发生文件界面图

Fig.6-8 Send file interface diagram

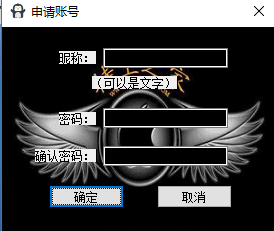


图6-9 注册界面图

Fig.6-9 Register interface diagram



图6-10 登陆界面图

Fig.6-10 Login interface diagram

## 6.4 程序清单

详见附录C。

# 7 系统测试

## 7.1 测试的目的

测试的主要目的是用尽可能低的成本和尽可能少的测试方案，尽可能多的发现系统错误。软件测试从不同的角度出发会派生出两种不同的测试原则，从用户的角度出发，就是希望通过软件测试能充分暴露软件中存在的问题和缺陷，从而考虑是否可以接受该产品。从开发者的角度出发，就是希望测试能表明软件产品不存在错误，已经正确地实现了用户的需求，确立人们对软件质量的信心。

测试在软件开发过程中一直都是不可或缺的，在传统的软件工程中，就有一个明确、独立的测试阶段。随着软件危机的频频出现以及人们对软件实质的进一步的认识，测试的地位得到了前所未有的提高。测试已经不仅仅局限于开发的一个阶段，它已经贯穿这个软件的开发过程，人们也已经意识到：测试开始的越早，测试执行的越频繁，不断降低整个软件开发成本。

## 7.2 测试方案

系统的测试方法主要使用以下两种：

一是黑盒测试：是通过使用整个软件或某种软件功能来严格地测试, 而并没有通过检查程序的源代码或者很清楚地了解该软件的源代码程序具体是怎样设计的。测试人员通过输入他们的数据然后看输出的结果从而了解软件怎样工作。黑盒测试是将程序看成不透明的，不去考虑其内部的结构和路径，只管程序的整体功能，即输入与输出数据转换是否和预期的结果一致。黑盒法发现程序中的错误，必须使用所有可能输入的数据来检查程序是否能得到正确的输出。黑盒测试着眼于程序外部结构，不考虑内部逻辑结构，主要针对软件界面和软件功能进行测试。

二是白盒测试：白盒测试与黑盒测试正好相反，是通过对程序的源代码进行测试而不使用用户界面。这种类型的测试需要从代码句法发现内部代码在算法，溢出，路径，条件等等中的缺点或者错误，进而加以修正。白盒测试是将程序看作一个透明的盒子，对于程序内部构造要十分了解，从而对程序内部的所有逻辑路径进行测试，故白盒法又称为逻辑覆盖。如果想用白盒法发现程序中的所有错误，则需要使程序中每种可能的路径都实际执行一次。

白盒测试主要是测试程序中是否有算法或者逻辑性错误，而黑河测试主要是测试系统的界面和功能是否有错误。本系统测试主要使用了白盒测试中的条件覆盖方法，以及黑盒测试的通过测试和失败测试。

## 7.3 测试结果

经过对软件进行测试，从发现问题到解决问题的反复修改，软件的界面显示、基本功能的使用已经完成，与预期结果相同。

# 8 结论

经过整个系统的设计和开发，这个局域网即时聊天软件已基本完成。主要能实现在局域网下用户之间文字、文件的发送和接受，以及登陆、注册、头像更改、宠物界面、更改字体、查询消息记录等功能。在程序中的设计的注册功能，给新用户的加入提供了入口，为软件的可用性提供了保障。本软件设计、界面、操作等设计都比较简洁，用户容易上手使用。为实现以上的许多功能，开发过程中遇到了许多困难，但经过向老师和开发人员咨询，查阅资料书籍等，最终还是坚持了下来。但是由于人力物力有限，该软件的部分功能与预期的还有一些差距，比如加入语音聊天、图形图像的发送、用户界面不够细腻等，这部分有待加强改善。

随着计算机网络的迅速发展，现实中的交流将会慢慢转移到网络虚拟交流之中，文中主要通过局域网内的聊天软件的开发，提高对计算机网络的认识。

# 致谢

我的毕业设计完成了，在此，我先要向指导老师彭晏飞老师表示我深深的感谢和崇高的敬意。在这次毕业设计从选题、设计到实现的过程中，彭老师给予我了极大帮助，尽管工作繁忙，但是他也能够抽出时间了解我的设计情况，并为我指点迷津，帮助我完成软件设计和开发。

感谢我的同学们，从遥远的家乡一同来到这个陌生的城市，共同学习、玩耍，是你们给我家人般的感觉，让我在这四年里不孤单，能够顺利完成学业之余，生活也变得多姿多彩，是他们与我一起走过大学的路，感谢一路上有你们的陪伴。

我还要感谢在大学这四年里给予我帮助和关怀老师和朋友，是他们丰富了我的大学生活，让我学到了许多课本没有的知识，让我成长的更加迅速。

最后还要最感谢父母，是他们养育了我，给了我生命，在人生的道路上为我领航。

# 参考文献

[1] 熊世桓.用Socket编程实现TCP/IP网络接口[J].贵州师范学院报,2003,14(4):86-90.

[2] 邓权良.Winsock网络程序设计[M].北京:中国铁道出版社,2002.

[3] 徐峰,陈暄.UML面向对象建模基础[M].北京:中国水利水电出版社,2006.

[4] 张海藩,牟永敏.软件工程导论[M].北京:清华大学出版社,2013.

[5] 谢钧,谢希仁.计算机网络教程[M].北京:人民邮电出版社,2014.

[6] 侯俊杰.深入浅出MFC[M]. 武汉:华中科技大学出版社,2001.

[7] 刘文建.浅谈如何加强计算机网络管理技术创新应用[J].企业文化(中旬刊),2016,12(5):262-262.

[8] David J.Kruglinski.Visual C++ 技术内幕[M].北京:清华大学出版社,1999.

[9] 梁普选.Visual C++ 序设计与实践.北京:清华大学出版社,2005.

[10] 严蔚敏,吴伟民.数据结构[M].北京:清华大学出版社:2007.

[11] Douglas E.Comer,David L.Stevens .Internetworking with TCP/IP [M].US:Prentice Hall,2000.

[12] Andrew S.Tanenbaum,David J.Wetherall .Computer Networks(5thEdition)[M].US:Prentice Hall,1998.

# 附录A 译文

抽象和类

生活中充满复杂性，处理复杂性的一个方法是简化和抽象。人的身体是由无数个原子组成的，而一些心理学者会说，人的思想是由半自助的主体组成的。但将人自己视为单一实体要简单得多。在计算中，抽象是在与用户的接口方面表示信息的关键步骤。也就是说，您将提出问题的基本操作特征，并以这些术语表达解决方案。在垒球统计示例中，界面描述了用户如何初始化，更新和显示数据。从抽象，它是用户定义类型的一小步，在C ++中是实现抽象接口的类设计。

什么是类型？

我们再多想一下什么是一种类型。例如，什么是讨厌鬼？受流行的固定模式影响，你可能会在视觉上想到一个讨厌鬼等特点：胖、戴黑色宽边眼镜，兜里插满钢笔等。经过一番思考之后，你可能会得出结论，一个讨厌鬼在运作上有更好的界定，例如，就如何回应一个尴尬的社会情况而言。你也有类似的情况，如果你不介意使用诸如C之类的过程语言进行延伸的类比。首先，你会倾向于根据外观来考虑一个数据类型，它是如何存储在内存中的。例如，char是1字节的内存，而double通常是8字节的内存。但是，有一点反映可以让您得出结论，数据类型也可以根据可执行的操作进行定义。例如，int类型可以使用所有的算术运算。你可以添加，减法，乘法和分割整数。你也可以使用模数运算符（％）。

另一方面，考虑指针。指针可能非常需要与int一样的内存。它甚至可以在内部表示为整数。但指针不允许与整数相同的操作。例如，你不能将两个指针相互相乘。这个概念没有任何意义，所以C ++没有实现它。因此，当你将变量声明为int或作为指针到浮点数时，你不仅仅是分配内存，你还要确定可以使用变量执行哪些操作。简而言之，指定一个基本类型有三件事情：

* 确定数据对象需要多少内存。
* 它确定内存中的位如何被解释（一个long和一个float可能在内存中使用相同的位数，但是它们被不同地转换为数值）。
* 确定可以使用数据对象执行哪些操作或方法。

对于内置类型，有关操作的信息内置于编译器中。但是，当你在C ++中定义用户定义的类型时，你必须自己提供相同类型的信息。为了更换这项额外的工作，你可以获得自定义的新数据类型的功能和灵活性，以符合现实世界的要求。

C ++中的类

一个类是用于将抽象转换为用户定义类型的C ++工具。它将数据表示和将数据操作到一个整齐的包中的方法进行组合。我们来看看一个代表股票的课程。

首先，你必须考虑一下如何代表股票。你可以将一股股票作为基本单位，并定义一个类来表示股票。但是，这意味着您需要100个对象来代表100个股份，这是不实际的。相反，你可以代表一个人目前持有的特定股票作为基本单位。拥有的股份数量将成为数据表示的一部分。一个现实的方法将必须保持对初始购买价格和购买日期的税务目的的记录。 而且它也必须管理诸如股票分割的事件。对于定义课程的第一个努力来说，这似乎有点野心勃勃，所以你可以采取理想化的，简化的事情观点。特别是，你可以将你可以执行的操作限制在以下方面：

* 在一家公司收购股票。
* 买入更多的股票。
* 卖出股票。
* 更新股票的每股价值。
* 显示有关持有的信息。

你可以使用此列表来定义股票类的公共接口（如果你有兴趣，你可以稍后添加其他功能。）要支持此界面，你需要存储一些信息。 再次，你可以使用简化的方法。 例如，不要担心美国以八分之一美元的倍数来评估股票的做法（显然纽约证券交易所必须在上一版本中看到这种简化，因为它已经决定转换为 这里使用的系统。）以下是要存储的信息列表：

* 公司名称
* 所有股数
* 每个股份的价值
* 所有股份总值

接下来，你可以定义类。一般来说，类规范有两部分：

* 在成员函数方面，根据数据成员和公共接口描述数据组件的类声明，称为方法
* 类方法定义，描述某些类成员函数的实现方式

大致来说，类声明提供了一个类概述，而该方法定义提供细节。

什么是接口？

接口是用于两个系统之间的交互的共享框架，例如在计算机和打印机之间，或者在用户和计算机程序之间。例如，用户可能是你，程序可能是文字处理器。当你使用文字处理器时，不要将单词直接传输到计算机内存。而是与程序提供的界面进行交互。您按下一个键，电脑会在屏幕上显示一个字符。移动鼠标，计算机将光标移动到屏幕上。你不小心点击鼠标，并且奇怪的是，你打字的段落。程序界面管理你的意图转换到计算机中存储的特定信息。

对于类，我们谈论公共接口。在这种情况下，公众是使用类的程序，交互系统由类对象组成，接口由编写该类的人提供的方法组成。该接口使程序员能够编写与类对象交互的代码，从而使程序能够使用类对象。例如，要查找字符串对象中的字符数，你不要将对象打开到内部的内容中；你只需使用类创建者提供的size（）方法。事实证明，类设计不能直接访问公共用户。但是公众可以使用size（）方法。然后，size（）方法是用户和字符串类对象之间的公共接口的一部分。类似地，getline（）方法是istream类public interface的一部分；使用cin的程序不直接修改cin对象的内部来读取一行输入；相反，getline（）做的工作。

如果你想要一个更加个人的关系，而不是使用一个类作为公共用户的程序，你可以考虑使用该类作为公共用户编写程序的人。但是无论如何，要使用一个类，你需要知道它的公共接口；要写一个类，你需要创建它的公共接口。

开发类和使用它的程序需要几个步骤。不要一下子把它们全部放在一起，而是把发展分解成更小的阶段。通常，C ++程序员将接口以类定义的形式放置在头文件中，并以源代码文件的类方法的代码形式放置了实现。 所以我们来代表一下。清单10.1显示了第一个阶段，一个名为库的类的暂定类声明。该文件使用#ifndef等，如第9章“内存模型和名称空间”中所述，以防止多个文件夹入。

稍后你会再看一下课程细节，但是首先来看一下更一般的功能。首先，C ++关键字类将清单10.1中的代码标识为定义类的设计（在这种情况下，关键字类和类型名称与模板参数的方式不同义; typename不能在这里使用。） 该语法标识库，作为这个新类的类型名称。 该声明使您能够声明库存类型的变量，称为对象或实例。 每个单独的对象代表一个单一的持有。 例如，以下声明创建两个名为sally和solly的Stock对象：

Stock sally;

Stock solly;

例如，sally的对象可以代表Sally在一家特定公司的股票。

接下来，请注意，您决定存储的信息以类数据成员的形式出现，例如公司和股份。例如，公司成员持有公司名称，股份成员持有股份数Sally拥有，share\_val成员持有每个股份的价值，total\_val成员持有所有股份的总价值。类似地，期望的操作显示为类函数成员（或方法），如sell（）和update（）。可以在某个位置定义成员函数，例如set\_tot（）或者可以由原型来表示，像这个类的其他成员函数一样。其他成员函数的完整定义稍后在实现文件中，但原型足以描述函数接口。将数据和方法绑定到单个单元中是该类最显着的特征。因为这个设计，创建一个库存对象会自动建立一个管理如何使用该对象的规则。

你已经了解了istream和ostream类如何具有成员函数，例如get（）和getline（）。库类声明中的函数原型演示了成员函数的建立方式。 iostream头文件例如在istream类声明中有一个getline（）原型。

访问控制

关键字私有和公共也是新的。这些标签描述了类成员的访问控制。任何使用特定类的对象的程序都可以直接访问公共部分。程序只能通过使用公共成员函数访问对象的私有成员（或者，你可以通过一个朋友函数在第11章“使用类”中看到）。例如，修改对象的唯一方法股票类别的股东是使用股票成员函数之一。因此，公共成员的职能作为程序和对象的私人成员之间的关系；它们提供对象和程序之间的接口。通过程序直接访问的数据绝缘称为数据隐藏（C ++提供了第三个访问控制关键字，受保护，当我们在第13章“继承”中讨论类继承时，我们将讨论它们）（）数据隐藏可能是股票基金招股说明书中的不道德行为，因为它保留了数据的完整性，这在计算中是一个很好的做法。

类设计试图将公共界面与实现细节分开。公共接口表示设计的抽象组件。将实现细节集中在一起并将其与抽象分离，称为封装。数据隐藏（将数据放入类的私有部分）是封装的一个实例，隐藏在私有部分中的实现的功能细节，因为库类使用set\_tot（）。另一个封装的例子是通常的 将类函数定义放在类声明的单独文件中的做法。

OOP和C ++

OOP是一种编程风格，你可以在某种程度上使用任何语言。当然，你可以将许多OOP的想法纳入普通的C程序。例如，第9章提供了一个示例（见清单9.1,9.2,9.3），其中头文件包含结构原型以及用于操作该结构的函数的原型。main（）函数只是定义该结构类型的变量，并使用相关函数来处理这些变量; main（）不直接访问结构成员。实质上，该示例定义了将存储格式和函数原型放在头文件中的抽象类型，从main（）隐藏实际的数据表示。 C ++包含专门用于实现OOP方法的功能，因此它使您能够使C进程比C更进一步。首先，将数据表示和函数原型放在一个单独的类声明中，而不是使它们分开统一将所有内容放在一个类声明中的描述。第二，使数据表示私有执行数据仅由授权功能访问的限制。如果在C示例中main（）直接访问结构成员，则它违反了OOP的精神，但不会破坏任何C语言规则。但是，试图直接访问Stock对象的共享成员，会破坏C ++语言规则，并且编译器将捕获它。

请注意，数据隐藏不仅可以阻止您直接访问数据，还可以免除你（作为类的用户），无需知道数据的显示方式。 例如，show（）成员显示了一个保持的总值。 该值可以作为对象的一部分存储，如清单10.1中的代码所示，或者可以在需要时进行计算。 从使用课堂的角度来看，使用哪种方法没有区别。 你需要知道的是不同的成员功能所完成的； 也就是说，你需要知道一个成员函数需要什么样的参数，它有什么样的返回值。原则是将实现的细节与界面的设计分开。 如果你以后找到实现数据表示或成员函数详细信息的更好方式，你可以在不更改程序界面的情况下更改这些细节，并使程序更容易维护。

成员访问控制：公共还是私有？

你可以在类的公共部分或私有部分声明类成员，无论它们是数据项还是成员函数。 但是，由于OOP的主要规则之一是隐藏数据，所以数据项通常进入私有部分。 构成类接口的成员函数进入公共部分； 否则，你无法从程序中调用这些函数。正如Stock声明所示，您还可以将成员函数放在私有部分中。你不能直接从程序中调用这些函数，但是公共方法可以使用它们。 通常，你使用私有成员函数来处理不构成公共接口一部分的实现细节。你不必在类声明中使用关键字private，因为它是类对象的默认访问控制：

class World

{

float mass; // private by default

char name[20]; // private by default

public:

void tellall(void);

...

};

然而，这本书明确地使用私有标签来强调数据隐藏的概念。

类和结构

类描述看起来很像结构声明，增加了成员函数和公共和私有可见性标签。事实上，C ++扩展到相同功能类的结构。唯一的区别是结构的默认访问类型是public，而类的默认类型是private。C ++程序员通常使用类实现类描述，同时将结构限制为表示纯数据对象（通常称为纯旧数据结构或POD结构）。

实现类成员函数

我们还必须创建类规范的第二部分：为类声明中的原型表示的成员函数提供代码。成员函数定义很像常规函数定义。每个都有一个函数头和一个函数体。成员函数定义可以有返回类型和参数。但他们也有两个特点：

* 定义成员函数时，使用范围解析运算符（::）来标识该函数所属的类。
* 类方法可以访问类的private组件。

首先，对于一个成员函数的函数报头使用范围分辨率算子（::），以指示该函数所属的类。例如，update（）成员函数的标题如下所示：

void Stock::update(double price)

这个符号表示法意味着我们正在定义作为Stock类的成员的update（）函数。这不仅将update（）标识为成员函数，这意味着您可以为不同类的成员函数使用相同的名称。例如，一个Buffoon类的update（）函数会有这个函数头：

void Buffoon::update()

因此，范围解析运算符解析方法定义适用的类的标识。我们说标识符update（）具有类范围。如果需要，库类的其他成员函数可以使用update（）方法，而不使用scope-resolution操作符。这是因为它们属于同一个类，在范围内使update（）。然而，在类声明和方法定义之外使用update（）需要特殊的措施，我们很快就会得到。

查看方法名的一种方法是类方法的完整名称包括类名。Stock :: update（）被称为函数的限定名称。另一方面，简单的update（）是一个全名的缩写（不合格的名称），只能在类范围内使用。方法的第二个特点是方法可以访问类的私有成员。例如，show（）方法可以使用如下代码：

std::cout << "Company: " << company

<< " Shares: " << shares << endl

<< " Share Price: $" << share\_val

<< " Total Worth: $" << total\_val << endl;

其中，company、shares等是Stock类的私人数据成员。如果你尝试使用非成员函数来访问这些数据成员，则编译器会禁止你这样做（但是第11章讨论的友元函数除外）。

成员函数说明

acquire（）函数管理给定公司的第一次股票收购，而buy（）和sell（）管理从现有控股增加或减去。 buy（）和sell（）方法确保购买或出售的股份数量不是负数。另外，如果用户尝试出售比他或她拥有的更多的股份，那么这个sell（）函数会终止交易。使数据私有并限制对公共功能的访问的技术可以控制数据的使用方式;在这种情况下，它允许您插入这些安全措施来防止错误的事务。

四个成员函数设置或重置total\_val成员值。该类不是将该计算写入四次，而是将每个函数调用set\_tot（）函数。因为这个函数只是实现代码而不是公共接口的一部分，所以这个类使得set\_tot（）是一个私有成员函数（也就是说，set\_tot（）是一个成员函数，用于编写类但不是用于编写使用该类的代码的人使用）。如果计算冗长，则可以节省一些打字和代码空间。然而，这里的主要价值在于，通过使用函数调用而不是每次重新输入计算，你都可以确保完全相同的计算完成。此外，如果你必须修改计算（在这种情况下不太可能），则必须在一个位置修改它。

# 附录B 外文文献

Abstraction and Classes

Life is full of complexities, and one way we cope with complexity is to frame simplifying

abstractions. You are a collection of more than an octillion atoms. Some students of the mind would say that your mind is a collection of several semiautonomous agents. But it’s much simpler to think of yourself as a single entity. In computing, abstraction is the crucial step of representing information in terms of its interface with the user. That is, youabstract the essential operational features of a problem and express a solution in those terms. In the softball statistics example, the interface describes how the user initializes, updates, and displays the data. From abstraction, it is a short step to the user-defined type, which in C++ is a class design that implements the abstract interface.

What Is a Type?

Let’s think a little more about what constitutes a type. For example, what is a nerd? If you subscribe to the popular stereotype, you might think of a nerd in visual terms—thick,

black-rimmed glasses, pocket protector full of pens, and so on. After a little reflection, you might conclude that a nerd is better defined operationally—for example, in terms of how he or she responds to an awkward social situation. You have a similar situation, if you don’t mind stretched analogies, with a procedural language such as C. At first, you tend to think of a data type in terms of its appearance—how it is stored in memory. A char ,for example, is 1 byte of memory, and a double is often 8 bytes of memory. But a little reflection leads you to conclude that a data type is also defined in terms of the operations that can be performed on it. For example, the int type can use all the arithmetic operations. You can add, subtract, multiply, and divide integers. You can also use the modulus operator( % ) with them.

On the other hand, consider pointers. A pointer might very well require the sameamount of memory as an int. It might even be represented internally as an integer. But a pointer doesn’t allow the same operations that an integer does. You can’t, for example, multiply two pointers by each other. The concept makes no sense, so C++ doesn’t implement it. Thus, when you declare a variable as an int or as a pointer-to- float ,you’re not just allocating memory—you are also establishing which operations can be performed with the variable. In short, specifying a basic type does three things:

* It determines how much memory is needed for a data object.
* It determines how the bits in memory are interpreted.(A long and a float might use the same number of bits in memory, but they are translated into numeric values differently.)
* It determines what operations, or methods, can be performed using the data object.

For built-in types, the information about operations is built in to the compiler. Butwhen you define a user-defined type in C++,you have to provide the same kind ofinformation yourself. In exchange for this extra work, you gain the power and flexibility to custom fit new data types to match real-world requirements.

Classes in C++

A class is a C++ vehicle for translating an abstraction to a user-defined type. It combines data representation and methods for manipulating that data into one neat package. Let’s look at a class that represents stocks.

First, you have to think a bit about how to represent stocks. You could take one share of stock as the basic unit and define a class to represent a share. However, that implies that you would need 100 objects to represent 100 shares, and that’s not practical. Instead, you can represent a person’s current holdings in a particular stock as a basic unit. The number of shares owned would be part of the data representation. A realistic approach would have to maintain records of such things as initial purchase price and date of purchase for tax purposes. Also it would have to manage events such as stock splits. That seems a bit ambitious for a first effort at defining a class, so you can instead take an idealized, simplified view of matters. In particular, you can limit the operations you can perform to the following:

* Acquire stock in a company.
* Buy more shares of the same stock.
* Sell stock.
* Update the per-share value of a stock.
* Display information about the holdings.

You can use this list to define the public interface for the stock class.(And you can add additional features later if you’re interested.) To support this interface, you need to store some information. Again, you can use a simplified approach. For example, don’t worry about the U.S. practice of evaluating stocks in multiples of eighths of a dollar.(Apparently the New York Stock Exchange must have seen this simplification in a previous edition of the book because it has decided to change over to the system used here.) Here’s a list of information to store:

* Name of company
* Number of stocks owned
* Value of each share
* Total value of all shares

Next, you can define the class. Generally, a class specification has two parts:

* A class declaration, which describes the data component, in terms of data members, and the public interface, in terms of member functions, termed methods
* The class method definitions, which describe how certain class member functions are implemented

Roughly speaking, the class declaration provides a class overview, whereas the method

definitions supply the details.

What Is an Interface?

An interface is a shared framework for interactions between two systems—for instance, between a computer and a printer or between a user and a computer program. For example, the user might be you and the program might be a word processor. When you use the word processor, you don’t transfer words directly from your mind to the computer memory. Instead, you interact with the interface provided by the program. You press a key, and the computer shows you a character on the screen. You move the mouse, and the computer moves a cursor on the screen. You click the mouse accidentally, and something weird hap-pens to the paragraph you were typing. The program interface manages the conversion of your intentions to specific information stored in the computer.

For classes, we speak of the public interface. In this case, the public is the program using the class, the interacting system consists of the class objects, and the interface consists of the methods provided by whoever wrote the class. The interface enables you, the programmer, to write code that interacts with class objects, and thus it enables the program to use the class objects. For example, to find the number of characters in a string object, you don’t open up the object to what is inside; you just use the size() method provided by the class creators. It turns out that the class design denies direct access to the public user. But the public is allowed to use the size() method. The size() method, then, is part of the public interface between the user and a string class object. Similarly, the getline() method is part of the istream class public interface; a program using cin doesn’t tinker directly with the innards of a cin object to read a line of input; instead, getline() does the work.

If you want a more personal relationship, instead of thinking of the program using a class as the public user, you can think of the person writing the program using the class as the public user. But in any case, to use a class, you need to know its public interface; to write a class, you need to create its public interface.

Developing a class and a program using it requires several steps. Rather than take them all at once, let’s break up the development into smaller stages. Typically, C++ programmers place the interface, in the form of a class definition, in a header file and place the imple mentation, in the form of code for the class methods, in a source code file. So let’s be typical. Listing 10.1 presents the first stage, a tentative class declaration for a class called Stock .The file uses #ifndef, and so on, as described in Chapter 9,“Memory Models and Name-spaces,” to protect against multiple file inclusions.

You’ll get a closer look at the class details later, but first let’s examine the more general features. To begin, the C++ keyword class identifies the code in Listing 10.1 as defining the design of a class.(In this context the keywords class and typename are not synonymous the way they were in template parameters; typename can’t be used here.) The syntax identifies Stock as the type name for this new class. This declaration enables you to declare variables, called objects, or instances, of the Stock type. Each individual object represents a single holding. For example, the following declarations create two Stock objects called sally and solly :

Stock sally;

Stock solly;

The sally object, for example, could represent Sally’s stock holdings in a particular company.

Next, notice that the information you decided to store appears in the form of class data members, such as company and shares .The company member of sally ,for example, holds the name of the company, the share member holds the number of shares Sally owns, the share\_val member holds the value of each share, and the total\_val member holds the total value of all the shares. Similarly, the desired operations appear as class function members (or methods),such as sell() and update() .A member function can be defined in place—for example, set\_tot() —or it can be represented by a prototype, like the other member functions in this class. The full definitions for the other member functions come later in the implementation file, but the prototypes suffice to describe the function interfaces. The binding of data and methods into a single unit is the most striking feature of the class. Because of this design, creating a Stock object automatically establishes the rules governing how that object can be used.

You’ve already seen how the istream and ostream classes have member functions, such as get() and getline() .The function prototypes in the Stock class declaration demonstrate how member functions are established. The iostream header file, for example, has a getline() prototype in the istream class declaration.

Access Control

Also new are the keywords private and public .These labels describe access control for class members. Any program that uses an object of a particular class can access the public portions directly. A program can access the private members of an object only by using the public member functions (or, as you’ll see in Chapter 11,“Working with Classes,” via a friend function).For example, the only way to alter the shares member of the Stock class is to use one of the Stock member functions. Thus, the public member functions act as go-betweens between a program and an object’s private members; they provide the interface between object and program. This insulation of data from direct access by a pro-gram is called data hiding.(C++ provides a third access-control keyword, protected, which we’ll discuss when we cover class inheritance in Chapter 13,“Class Inheritance.”) (See Figure 10.1.) Whereas data hiding may be an unscrupulous act in, say, a stock fund prospectus, it’s a good practice in computing because it preserves the integrity of the data.

A class design attempts to separate the public interface from the specifics of the imple- mentation. The public interface represents the abstraction component of the design. Gathering the implementation details together and separating them from the abstraction is called encapsulation. Data hiding (putting data into the private section of a class) is an instance of encapsulation, and so is hiding functional details of an implementation in the private section, as the Stock class does with set\_tot() .Another example of encapsulation is the usual practice of placing class function definitions in a separate file from the class declaration.

OOP and C++

OOP is a programming style that you can use to some degree with any language. Certainly, you can incorporate many OOP ideas into ordinary C programs. For example, Chapter 9 provides an example (see Listings 9.1, 9.2, 9.3) in which a header file contains a structure prototype along with the prototypes for functions to manipulate that structure. The main() function simply defines variables of that structure type and uses the associated functions to handle those variables; main() does not directly access structure members. In essence, that example defines an abstract type that places the storage format and the function prototypes in a header file, hiding the actual data representation from main() . C++ includes features specifically intended to implement the OOP approach, so it enables you to take the process a few steps further than you can with C. First, placing the data representation and the function prototypes into a single class declaration instead of keeping them separate unifies the description by placing everything in one class declaration. Second, making the data representation private enforces the stricture that data is accessed only by authorized functions. If in the C example main() directly accesses a structure member, it violates the spirit of OOP, but it doesn’t break any C language rules. However, trying to directly access, say, the shares member of a Stock object does break a C++ language rule, and the compiler will catch it.

Note that data hiding not only prevents you from accessing data directly, but it also absolves you (in the roll as a user of the class) from needing to know how the data is represented. For example, the show() member displays, among other things, the total value of a holding. This value can be stored as part of an object, as the code in Listing 10.1 does, or it can be calculated when needed. From the standpoint of using the class, it makes no difference which approach is used. What you do need to know is what the different member functions accomplish; that is, you need to know what kinds of arguments a member function takes and what kind of return value it has. The principle is to separate the details of the implementation from the design of the interface. If you later find a better way to implement the data representation or the details of the member functions, you can change those details without changing the program interface, and that makes programs much easier to maintain.

Member Access Control: Public or Private?

You can declare class members, whether they are data items or member functions, either in the public or the private section of a class. But because one of the main precepts of OOP is to hide the data, data items normally go into the private section. The member functions that constitute the class interface go into the public section; otherwise, you can’t call those functions from a program. As the Stock declaration shows, you can also put member functions in the private section. You can’t call such functions directly from a program, but the public methods can use them. Typically, you use private member functions to handle implementation details that don’t form part of the public interface. You don’t have to use the keyword private in class declarations because that is the default access control for class objects:

class World

{

float mass; // private by default

char name[20]; // private by default

public:

void tellall(void);

...

};

However, this book explicitly uses the private label in order to emphasize the concept of data hiding.

Classes and Structures

Class descriptions look much like structure declarations with the addition of member functions and the public and private visibility labels. In fact, C++ extends to structures the same features classes have. The only difference is that the default access type for a structure is public , whereas the default type for a class is private . C++ programmers commonly use classes to implement class descriptions while restricting structures to representing pure data objects (often called plain-old data structures, or POD structures).

Implementing Class Member Functions

We still have to create the second part of the class specification: providing code for those member functions represented by a prototype in the class declaration. Member function definitions are much like regular function definitions. Each has a function header and a function body. Member function definitions can have return types and arguments. But they also have two special characteristics:

* When you define a member function, you use the scope-resolution operator ( :: ) to identify the class to which the function belongs.
* Class methods can access the private components of the class.

Let’s look at these points now.

First, the function header for a member function uses the scope-resolution operator ( :: ) to indicate to which class the function belongs. For example, the header for the update() member function looks like this:

void Stock::update(double price)

This notation means you are defining the update() function that is a member of the Stock class. Not only does this identify update() as a member function, it means you can use the same name for a member function for a different class. For example, an update() function for a Buffoon class would have this function header:

void Buffoon::update()

Thus, the scope-resolution operator resolves the identity of the class to which a method definition applies. We say that the identifier update() has class scope. Other member functions of the Stock class can, if necessary, use the update() method without using the scope-resolution operator. That’s because they belong to the same class, making update() in scope. Using update() outside the class declaration and method definitions, however, requires special measures, which we’ll get to soon.

One way of looking at method names is that the complete name of a class method includes the class name. Stock::update() is called the qualified name of the function. A simple update() ,on the other hand, is an abbreviation (the unqualified name) for the full name—one that can be used just in class scope. The second special characteristic of methods is that a method can access the private members of a class. For example, the show() method can use code like this:

std::cout << "Company: " << company

<< " Shares: " << shares << endl

<< " Share Price: $" << share\_val

<< " Total Worth: $" << total\_val << endl;

Here company , shares ,and so on are private data members of the Stock class. If you try to use a nonmember function to access these data members, the compiler stops you cold in your tracks.(However, friend functions, which Chapter 11 discusses, provide an exception.)

With these two points in mind, we can implement the class methods as shown in Listing 10.2.We’ve placed them in a separate implementation file, so the file needs to include the stock00.h header file so that compiler can access the class definition. To provide more namespace experience, the code uses the std::qualifier in some methods and using declarations in others.

Member Function Notes

The acquire() function manages the first acquisition of stock for a given company, whereas buy() and sell() manage adding to or subtracting from an existing holding. The buy() and sell() methods make sure that the number of shares bought or sold is not a negative number. Also if the user attempts to sell more shares than he or she has, the sell() function terminates the transaction. The technique of making the data private and limiting access to public functions gives you control over how the data can be used; in this case, it allows you to insert these safeguards against faulty transactions.

Four of the member functions set or reset the total\_val member value. Rather than write this calculation four times, the class has each function call the set\_tot() function. Because this function is merely the means of implementing the code and not part of the public interface, the class makes set\_tot() a private member function.(That is, set\_tot() is a member function used by the person writing the class but not used by someone writing code that uses the class.) If the calculation were lengthy, this could save some typing and code space. Here, however, the main value is that by using a function call instead of retyping the calculation each time, you ensure that exactly the same calculation gets done. Also if you have to revise the calculation (which is not likely in this particular case),you have to revise it in just one location.

# 附录C 代码清单

1 服务器

（1）头文件：

typedef struct \_SOCKET\_STREAM\_FILE\_INFO {

*TCHAR* szFileTitle[128]; //文件的标题名

*DWORD* dwFileAttributes; //文件的属性

*FILETIME* ftCreationTime; //文件的创建时间

*FILETIME* ftLastAccessTime; //文件的最后访问时间

*FILETIME* ftLastWriteTime; //文件的最后修改时间

*DWORD* nFileSizeHigh; //文件大小的高位双字

*DWORD* nFileSizeLow; //文件大小的低位双字

*DWORD* dwReserved0; //保留，为0

*DWORD* dwReserved1; //保留，为0

}SOCKET\_STREAM\_FILE\_INFO, \*PSOCKET\_STREAM\_FILE\_INFO;

class CTcpSocketClient : public *CSocket*

{

public:

CTcpSocketClient(*CPtrList*\* pList);

virtual ~CTcpSocketClient();

virtual void OnClose(int nErrorCode);

virtual void OnReceive(int nErrorCode);

public:

*CPtrList*\* m\_pList;//保存服务器Socket中List的东西

*CString* m\_strID; //连接账号

*CString* m\_strIDtemp;

*CString* m\_strIp;//连接IP

void SendMsg(CHead Msg);

void SendFile(CHead Msg);

};

（2）源文件：

void CTcpSocketClient::OnReceive(int nErrorCode)

{

// TODO: 在此添加专用代码和/或调用基类

int n,tag = 0;

CHead Msg;

n=*Receive*(&Msg,sizeof(Msg));

if(Msg.type == 1)//登录信息

{

m\_strID = Msg.usermsg.ID;

m\_strIp = Msg.ip;

theApp.m\_clientMessage.*insert*(*make\_pair*(m\_strID, m\_strIp));

theApp.head.type = 1;

*SendMessage*(theApp.active->*GetSafeHwnd*(),WM\_MSG,0,0);

}

if(Msg.type == 4)//验证信息

{

m\_strID = Msg.usermsg.ID;

m\_strIp = Msg.ip;

for(int i=0;i!=theApp.m\_UserMsg.*size*();i++)

{

if(*strcmp*(theApp.m\_UserMsg[i].ID,Msg.usermsg.ID) == 0&&*strcmp*(theApp.m\_UserMsg[i].PASSWORD,Msg.usermsg.PASSWORD) == 0)

{

tag = 1;

*strcpy\_s*(Msg.usermsg.NAME,theApp.m\_UserMsg[i].NAME);

break;

}

}

if(tag != 1)

Msg.tag = 0;

}

if(Msg.type == 5)//注册信息

{

m\_strIDtemp = Msg.msg;

m\_strIp = Msg.ip;

*CString* IDtemp;

int tagtemp = 0;

while(tagtemp == 0)

{

IDtemp.*Format*(L"%d",*rand*()%90000+10000);

int temp = 1;

for(int i=0;i!=theApp.m\_UserMsg.*size*();i++)

{

if(*strcmp*((*CStringA*)IDtemp,theApp.m\_UserMsg[i].ID) == 0)

{

temp = 0;

break;

}

}

if(temp == 1)

{

*strcpy\_s*(Msg.usermsg.ID,(*CStringA*)IDtemp);

theApp.m\_UserMsg.*push\_back*(Msg.usermsg);

tagtemp = 1;

}

else

tagtemp = 0;

}

}

SendMsg(Msg);

*CSocket*::*OnReceive*(nErrorCode);

}

void CTcpSocketClient::SendMsg(CHead Msg)

{

*POSITION* ps = m\_pList->*GetHeadPosition*(); //取得，所有用户的队列

switch(Msg.type)

{

case 0://有用户退出

case 1://有用户上线

while(ps!=*NULL*)

{

CTcpSocketClient\* pTemp = (CTcpSocketClient\*)m\_pList->*GetNext*(ps);

pTemp->*Send*(&Msg,sizeof(Msg));

}

break;

case 2://已在线用户告知刚上线用户我在线

case 3://发送消息

while(ps!=*NULL*)

{

CTcpSocketClient\* pTemp = (CTcpSocketClient\*)m\_pList->*GetNext*(ps);

if(pTemp->m\_strID == Msg.toID)

pTemp->*Send*(&Msg,sizeof(Msg));

}

break;

case 4:

while(ps!=*NULL*)

{

CTcpSocketClient\* pTemp = (CTcpSocketClient\*)m\_pList->*GetNext*(ps);

if(pTemp->m\_strID == Msg.fromID)

pTemp->*Send*(&Msg,sizeof(Msg));

}

break;

case 5:

while(ps!=*NULL*)

{

CTcpSocketClient\* pTemp = (CTcpSocketClient\*)m\_pList->*GetNext*(ps);

if(pTemp->m\_strIDtemp == Msg.msg)

{

pTemp->*Send*(&Msg,sizeof(Msg));

pTemp->m\_strIDtemp = L"";

}

}

break;

case 6: //文件传输

SendFile(Msg);

break;

default:

break;

}

}

void CTcpSocketClient::SendFile(CHead Msg)

{

CTcpSocketClient\* pTemp = *NULL*;

*POSITION* ps = m\_pList->*GetHeadPosition*(); //取得，所有用户的队列

while (ps != *NULL*)

{

pTemp = (CTcpSocketClient\*)m\_pList->*GetNext*(ps);

//pTemp->m\_strID == Msg.fromID;

if (pTemp->m\_strID == Msg.toID)

{

pTemp->*Send*(&Msg, sizeof(Msg));

break;

}

}

if (pTemp == *NULL*)

{

return;

}

SOCKET\_STREAM\_FILE\_INFO StreamFileInfo;

*Receive*(&StreamFileInfo, sizeof(SOCKET\_STREAM\_FILE\_INFO));

*CFile* destFile(StreamFileInfo.szFileTitle, *CFile*::*modeCreate* | *CFile*::*modeWrite* | *CFile*::*typeBinary* | *CFile*::*shareDenyNone*);

*UINT* dwRead = 0;

while (dwRead<StreamFileInfo.nFileSizeLow)

{

*byte*\* data = new *byte*[1024];

*memset*(data, 0, 1024);

*UINT* dw = *Receive*(data, 1024);

destFile.*Write*(data, dw);

dwRead += dw;

}

*SetFileTime*((*HANDLE*)destFile.*m\_hFile*, &StreamFileInfo.ftCreationTime,

&StreamFileInfo.ftLastAccessTime, &StreamFileInfo.ftLastWriteTime);

*SetFileAttributes*(StreamFileInfo.szFileTitle,StreamFileInfo.dwFileAttributes);

destFile.*Close*();

//接受完成后转发至目标客户中

*CFile* file;

if (!file.*Open*(StreamFileInfo.szFileTitle, *CFile*::*modeRead*))

{

*AfxMessageBox*(L"无指定文件");

return;

}

SOCKET\_STREAM\_FILE\_INFO StreamFileInfo1;

*WIN32\_FIND\_DATA* FindFileData;

*FindClose*(*FindFirstFile*(StreamFileInfo.szFileTitle, &FindFileData));

*memset*(&StreamFileInfo1, 0, sizeof(SOCKET\_STREAM\_FILE\_INFO));

//StreamFileInfo.szFileTitle = myFile.GetFileTitle();

wsprintf(StreamFileInfo1.szFileTitle, L"%s", file.*GetFileTitle*());

//strcpy\_s(StreamFileInfo.szFileTitle, myFile.GetFileTitle());

int a = 0;

StreamFileInfo1.dwFileAttributes = FindFileData.*dwFileAttributes*;

StreamFileInfo1.ftCreationTime = FindFileData.*ftCreationTime*;

StreamFileInfo1.ftLastAccessTime = FindFileData.*ftLastAccessTime*;

StreamFileInfo1.ftLastWriteTime = FindFileData.*ftLastWriteTime*;

StreamFileInfo1.nFileSizeHigh = FindFileData.*nFileSizeHigh*;

StreamFileInfo1.nFileSizeLow = FindFileData.*nFileSizeLow*; pTemp->*Send*(&StreamFileInfo1, sizeof(SOCKET\_STREAM\_FILE\_INFO));

dwRead = 0;

while (dwRead<StreamFileInfo1.nFileSizeLow)

{

*byte*\* data = new *byte*[1024];

*UINT* dw = file.*Read*(data, 1024);

pTemp->*Send*(data, dw);

dwRead += dw;

}

file.*Close*();

}

2 客户端

（1）头文件

typedef struct \_SOCKET\_STREAM\_FILE\_INFO {

TCHAR szFileTitle[128]; //文件的标题名

DWORD dwFileAttributes; //文件的属性

FILETIME ftCreationTime; //文件的创建时间

FILETIME ftLastAccessTime; //文件的最后访问时间

FILETIME ftLastWriteTime; //文件的最后修改时间

DWORD nFileSizeHigh; //文件大小的高位双字

DWORD nFileSizeLow; //文件大小的低位双字

DWORD dwReserved0; //保留，为0

DWORD dwReserved1; //保留，为0

} SOCKET\_STREAM\_FILE\_INFO, \*PSOCKET\_STREAM\_FILE\_INFO;

class CClientApp : public CWinApp

{

public:

CFont m\_font; //字体

CClientApp();

CHead head;

CClientDlg \*active;

CLogin \*log;

CUdpSocket \*m\_udp;

CTcpSocket \*m\_tcp;

int search;//服务器搜寻标识

int tag;//账号密码匹配标识

CLogin login;//登录菜单

CString serverip;//记录服务器IP

CString m\_localip;//本地IP

CString m\_userID;//账号

CString m\_userNAME;//用户名

map<CString,CChat \*> m\_userMsg;//记录聊天对话框

map<CString,CString> m\_MsgRecode;//记录聊天信息

map<CString,CString> m\_clientMsg;//记录在线客户端ID == ip

map<CString,CString> m\_client;//ID == NAME

private:

void GetLocalIp(CString &ip);

// 重写

public:

virtual BOOL InitInstance();

// 实现

DECLARE\_MESSAGE\_MAP()

virtual int ExitInstance();

};

extern CClientApp theApp; //全局变量存放信息

（2）源文件

void CTcpSocket::OnReceive(int nErrorCode)

{

// TODO: 在此添加专用代码和/或调用基类

int n;

CString temp;

CHead Msg;

n=Receive(&Msg,sizeof(Msg));

switch(Msg.type)

{

case 0:

case 1:

case 2:

UpdateListCtrl(Msg);

break;

case 3:

ChatUpdateMsg(Msg);

break;

case 4: ////把用户信息发送给服务器

if(Msg.tag != 0)

{

theApp.tag = 1;

theApp.m\_userNAME = Msg.usermsg.NAME;

}

SendMessage(theApp.log->GetSafeHwnd(),WM\_MSG,0,0);

break;

case 5:

theApp.m\_userID = Msg.usermsg.ID;

temp += L"恭喜您注册成功，账号是：" + theApp.m\_userID + L"，请牢记！";

theApp.log->m\_userID = Msg.usermsg.ID;

theApp.log->UpdateData(false);

AfxMessageBox(temp);

break;

case 6://文件传输

ReceiveFile();

break;

default:

break;

}

CSocket::OnReceive(nErrorCode);

}

void CTcpSocket::ReceiveFile()

{

CString fileName;

SOCKET\_STREAM\_FILE\_INFO StreamFileInfo;

theApp.m\_tcp->Receive(&StreamFileInfo, sizeof(SOCKET\_STREAM\_FILE\_INFO));

CFileDialog saveDlg(false, NULL, StreamFileInfo.szFileTitle, OFN\_HIDEREADONLY | OFN\_OVERWRITEPROMPT, NULL, NULL, 0);

if (saveDlg.DoModal() == IDOK)

{

fileName = saveDlg.GetPathName();

}

else

{

//closesocket(sock);

return;

}

//recvFileBuf[SIZEFILE]={0};

CFile destFile(fileName, CFile::modeCreate | CFile::modeWrite | CFile::typeBinary);

//CFile destFile(StreamFileInfo.szFileTitle, CFile::modeCreate | CFile::modeWrite | CFile::typeBinary);

UINT dwRead = 0;

while (dwRead<StreamFileInfo.nFileSizeLow)

{

byte\* data = new byte[1024];

memset(data, 0, 1024);

UINT dw = Receive(data, 1024);

destFile.Write(data, dw);

dwRead += dw;

}

SetFileTime((HANDLE)destFile.m\_hFile, &StreamFileInfo.ftCreationTime,

&StreamFileInfo.ftLastAccessTime, &StreamFileInfo.ftLastWriteTime);

SetFileAttributes(StreamFileInfo.szFileTitle, StreamFileInfo.dwFileAttributes);

destFile.Close();

CString temp = (CString)theApp.head.fromID + L"对你发送文件，请查收！";

AfxMessageBox(temp);

}

void CTcpSocket::ChatUpdateMsg(CHead Msg)//接收消息

{

CTime time;

time = CTime::GetCurrentTime(); //获取现在时间

CString strTime = time.Format(L"%Y-%m-%d %H:%M:%S ");

map<CString,CChat\*>::iterator it = theApp.m\_userMsg.find((CString)Msg.fromID);

CString temp;

temp = L"(" + (CString)Msg.fromID + L") " + (CString)Msg.usermsg.NAME;

if(it != theApp.m\_userMsg.end())

{

//是否已与消息来源打开了聊天框，如果已经打开了，则处理消息；如果没有，则先打开对话框

it->second->m\_receiveMsg += strTime + temp + L"\r\n " + (CString)Msg.msg + L"\r\n";

theApp.m\_MsgRecode[(CString)Msg.fromID] += strTime + temp + L"\r\n " + (CString)Msg.msg + L"\r\n";

it->second->UpdateData(false);

it->second->m\_receiveCtrl.LineScroll(it->second->m\_receiveCtrl.GetLineCount()-1);

}

else

{

CChat\* c;

c = new CChat;

c->m\_toID = Msg.fromID;

c->m\_name = Msg.usermsg.NAME;

c->Create(IDD\_CHAT,theApp.active);

c->ShowWindow(SW\_SHOW);

/\*CString strTemp;

strTemp.Format(L"%s",Msg.fromID);\*/

theApp.m\_userMsg.insert(make\_pair(Msg.fromID,c));

/\*消息记录\*/

map<CString, CString>::iterator map\_it = theApp.m\_MsgRecode.find((CString)Msg.fromID);

if (map\_it == theApp.m\_MsgRecode.end())

{

theApp.m\_MsgRecode.insert(make\_pair(Msg.fromID, L""));

}

else

{

c->m\_receiveMsg += theApp.m\_MsgRecode[(CString)Msg.fromID];

}

c->m\_receiveMsg += strTime + temp + L"\r\n " + (CString)Msg.msg + L"\r\n";

theApp.m\_MsgRecode[(CString)Msg.fromID] += strTime + temp + L"\r\n " + (CString)Msg.msg + L"\r\n";

c->UpdateData(false);

c->m\_receiveCtrl.LineScroll(c->m\_receiveCtrl.GetLineCount()-1);

}

}

void CTcpSocket::UpdateListCtrl(CHead Msg)//更新在线用户列表

{

if(Msg.type == 0)//有客户端退出

{

theApp.m\_clientMsg.erase((CString)Msg.usermsg.ID);

theApp.m\_client.erase((CString)Msg.usermsg.ID);

}

else if(Msg.type == 1)//有客户端加入

{

theApp.m\_clientMsg.insert(make\_pair((CString)Msg.usermsg.ID,(CString)Msg.ip));

theApp.m\_client.insert(make\_pair((CString)Msg.usermsg.ID,(CString)Msg.usermsg.NAME));

CHead temp;

temp.type = 2;

strcpy\_s(temp.fromID,(CStringA)theApp.m\_userID);

strcpy\_s(temp.toID,Msg.fromID);

strcpy\_s(temp.usermsg.NAME,(CStringA)theApp.m\_userNAME);

strcpy\_s(temp.ip,(CStringA)theApp.m\_localip);

strcpy\_s(temp.usermsg.ID,(CStringA)theApp.m\_userID);

theApp.m\_tcp->Send(&temp,sizeof(temp));

}

else if(Msg.type == 2)//我已在线……

{

theApp.m\_clientMsg.insert(make\_pair((CString)Msg.usermsg.ID,(CString)Msg.ip));

theApp.m\_client.insert(make\_pair((CString)Msg.usermsg.ID,(CString)Msg.usermsg.NAME));

}

map<CString,CString>::iterator it;

int i;

theApp.active->m\_listcontrol.DeleteAllItems();

for(it = theApp.m\_clientMsg.begin(),i = 0;it != theApp.m\_clientMsg.end();it++,i++)

{

theApp.active->m\_listcontrol.InsertItem(i, it->first);

theApp.active->m\_listcontrol.SetItemText(i,1,theApp.m\_client.find(it->first)->second);

theApp.active->m\_listcontrol.SetItemText(i, 2, it->second);

}

}