**目录**

[**1** **实现原理** 1](#_Toc484113940)

[**2** **程序流程图** 1](#_Toc484113941)

[**3** **程序设计** 3](#_Toc484113942)

[**3.1** **消息发送格式** 3](#_Toc484113943)

[**3.1.1** **基本格式：*类别\*内容*** 3](#_Toc484113944)

[**3.1.2** **类别说明：** 3](#_Toc484113945)

[**3.2** **服务器界面设计** 4](#_Toc484113946)

[**3.3** **客户端界面设计** 5](#_Toc484113947)

[**4** **代码清单** 6](#_Toc484113948)

[**4.1** **程序头文件** 6](#_Toc484113949)

[**4.2** **服务器端代码** 7](#_Toc484113950)

[**4.3** **客户端代码** 19](#_Toc484113951)

1. **实现原理**

如今的一些聊天软件（如：QQ）实现通信的模式主要有两种，一种一对多模式，即一个服务器对多个客户端。一种是多对多模式，即客户端带有服务端的一些功能，能与客户端之间直接建立连接通信。本程序采用C#中Socket类来实现。程序分为服务器端和客户端，采用TCP协议建立客户端与服务器之间的通信。服务器不断监听客户端的接入和客户端发送的消息。群发消息由服务器转发给所有连接上服务器的客户端，而一对一的私聊也由服务器转发给指定客户端。

1. **程序流程图**



**服务器运作过程**



**客户端运作过程**

1. **程序设计**
   1. **消息发送格式**
      1. **基本格式：*类别\*内容***
      2. **类别说明：**

由于服务器和客户端还需要辨别用户名信息、上线信息、下线信息、和在线客户的所有列表信息，所以采用了一些数字类别来规定信息的类别加在发送信息的前面。

**表1 消息分类规定表**

|  |  |  |
| --- | --- | --- |
| 标识号 | 服务器解释 | 客户端解释 |
| 1 | 有新用户接入连接的消息 | 有其他用户上线的消息 |
| 2 | 用户提出获取在线所有客户列表（仅在有新用户开始登录时会接收到） | 在线所有用户列表信息（仅在用户刚登陆时申请） |
| 3 | 用户退出登录的消息 | 有其他用户下线的消息 |
| 4 | 用户发送的群聊消息 | 收到了一条群聊消息 |
| 5 | 用户发送的私聊消息 | 收到了一条私聊消息 |
| 6 | 不接收只发送此类消息 | 来自管理员的广播消息 |
| 7 | 不接收只发送此类消息 | 服务器关闭聊天服务的通知（接收后用户自动退出程序） |
| 8 | 不接收只发送此类消息 | 确认自己退出登录的通知（接受到服务器确认后再退出程序） |

* + 1. **内容说明：**

**表2 内容说明表**

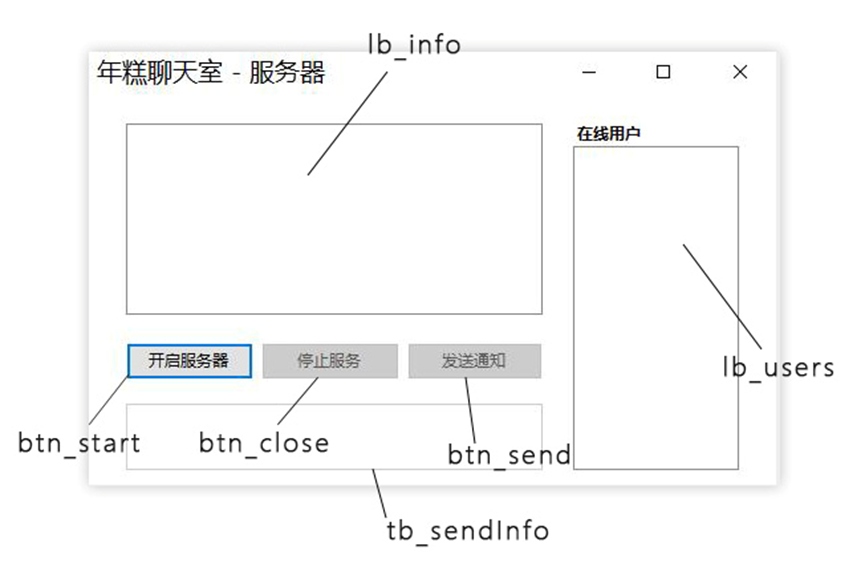
|  |  |  |
| --- | --- | --- |
| 类别号 | 格式 | 说明 |
| 1 | 用户名 | 按\*分割取类别以外剩下内容 |
| 2 | 用户1的用户名|用户1的IP地址|用户1的端口号&… | 按\*取分类外号剩下内容，再按&分割得到一个列表数组 |
| 3 |  | 服务器直接根据IP和端口定位socket |
| 4 | 用户名\*消息内容 | 按\*取下分类号和用户名，剩下部分为消息内容 |
| 5 | 用户1\*用户2|用户2的IP|用户2的端口号\*消息内容 | 按\*取下分类号和用户1和用户2，剩下部分为消息内容 |
| 6 | 消息内容 | / |
| 7 |  | / |
| 8 |  | / |

* 1. **服务器界面设计**

对服务器控件及功能的解释

1. lb\_info：listbox控件，用于显示消息。
2. lb\_users：listbox控件，用于显示在线用户列表。
3. tb\_sendInfo：textbox控件，用于管理员输入通知内容以待发送。
4. btn\_start：button按钮控件，用于启动聊天服务。
5. btn\_close：button按钮控件，用于关闭聊天服务。
6. btn\_send：butoon按钮空间，用于管理员向用户发送通知。

**图1 服务器界面及控件标识**

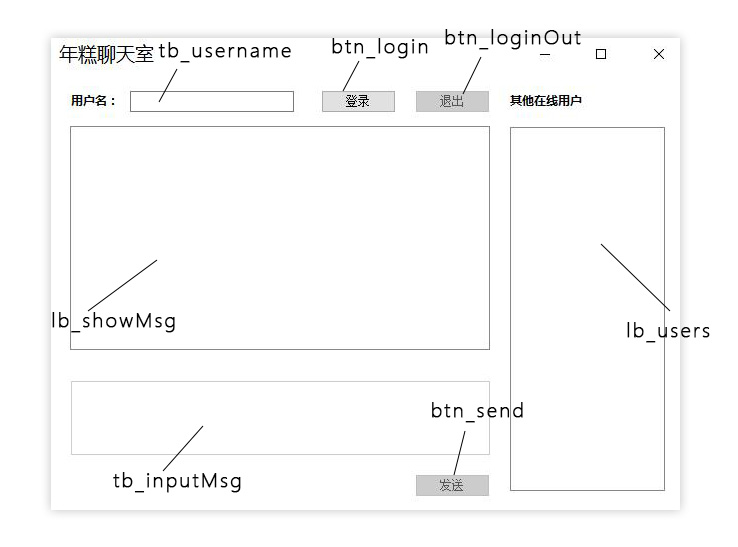


* 1. **客户端界面设计**

对服务器控件及功能的解释

1. tb\_username：textbox控件，让用户输入用户名。
2. tb\_login：button按钮控件，点击即可登录程序。
3. btn\_loginout：button按钮控件，用于用户退出登陆程序。
4. lb\_showMsg：listbox控件，用于显示收到的消息。
5. lb\_users：listbox控件，用于显示其他在线用户列表，可选择单聊对象。
6. tb\_inputMsg：textbox控件，用于输入发送消息。
7. btn\_send：button控件：用于发送消息。

**图2 客户端界面及控件标识**



1. **代码清单**
   1. **程序头文件**

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using System.Net.Sockets;

using System.Net;

using System.Threading;

using System.IO;

* 1. **服务器端代码**

namespace NGchatServer

{

public partial class Form1 : Form

{

//初始化窗体

public Form1()

{

InitializeComponent();

CheckForIllegalCrossThreadCalls = false;

setRestStatus();

}

Thread threadWatch = null;

Socket socketWatch = null;

Socket socConnection = null;

bool isntStop = true;

//创建字典收录所有客户端

Dictionary<string, Socket> dicSocket = new Dictionary<string, Socket>();

//开启服务

private void btn\_start\_Click(object sender, EventArgs e)

{

try

{

//开启监听端口

socketWatch = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);

IPAddress ipaddress = IPAddress.Parse("127.0.0.1");

IPEndPoint endpoint = new IPEndPoint(ipaddress, 7788);

//Socket绑定端口且开始监听连接

socketWatch.Bind(endpoint);

socketWatch.Listen(20);

//创建监听线程

threadWatch = new Thread(watchConnecting);

threadWatch.IsBackground = true;

threadWatch.Start();

lb\_info.Items.Add("聊天室服务已经开启啦");

setWorkStatus();

isntStop = true;

}

catch (Exception err)

{

MessageBox.Show(err.Message.ToString());

}

}

//创建监听函数

public void watchConnecting()

{

try

{

while (isntStop)

{

socConnection = socketWatch.Accept();

//创建通信线程

ParameterizedThreadStart pts = new ParameterizedThreadStart(serverRecMsg);

Thread thr = new Thread(pts);

thr.IsBackground = true;

thr.Start(socConnection);

}

}

catch

{

}

}

//服务器接收信息

public void serverRecMsg(object socketClientPara)

{

try

{

Socket socketServer = socketClientPara as Socket;

while (isntStop)

{

//创建一个对象保存获得消息

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

int length = socketServer.Receive(arrServerRecMsg);

string strMsg = Encoding.UTF8.GetString(arrServerRecMsg, 0, length);

string typeMsg = strMsg.Split('\*')[0];

string info = strMsg.Substring(2);

string user = "";

switch (typeMsg)

{

//其他用户上线的通知

case "1":

serverRecList(socketServer);

serverRecNewLogin(socketServer, info);

break;

//获取用户在线列表

case "2":

string[] list = info.Split('&');

for (int i = 0; i < list.Length - 1; i++)

{

lb\_users.Items.Add(list[i]);

}

break;

//其他用户退出的通知

case "3":

exitChat(socketServer);

break;

//收到群聊请求消息

case "4":

user = strMsg.Split('\*')[1];

info = info.Substring(user.Length + 1);

foreach (var item in dicSocket)

{

string sendMsg = "4\*" + user + "说：" + info;

byte[] arrSendMsg = Encoding.UTF8.GetBytes(sendMsg);

item.Value.Send(arrSendMsg);

}

lb\_info.Items.Add(user + "说：" + info);

lb\_info.SelectedIndex = lb\_info.Items.Count - 1;

break;

//收到单聊请求消息

case "5":

user = strMsg.Split('\*')[1];

string other = strMsg.Split('\*')[2];

info = info.Substring(user.Length + +other.Length + 2);

foreach (var item in dicSocket)

{

if (item.Key == other)

{

string sendMsg = "5\*" + user + " 对你说：" + info;

byte[] arrSendMsg = Encoding.UTF8.GetBytes(sendMsg);

item.Value.Send(arrSendMsg);

break;

}

}

break;

}

}

}

catch

{

}

}

//获取用户信息

public void serverRecNewLogin(object socketClientPara, string info)

{

try

{

IPAddress clientIP;

int clientPort;

//获取客户端IP

clientIP = (socConnection.RemoteEndPoint as IPEndPoint).Address;

//获取客户端端口

clientPort = (socConnection.RemoteEndPoint as IPEndPoint).Port;

string sendMsg = "1\*" + info + "|" + clientIP + "|" + clientPort;

byte[] arrSendMsg = Encoding.UTF8.GetBytes(sendMsg);

//发送所有在线用户新用户的上线通知

foreach (var item in dicSocket)

{

item.Value.Send(arrSendMsg);

}

lb\_users.Items.Add(info + "|" + clientIP + "|" + clientPort);

dicSocket.Add(info + "|" + clientIP + "|" + clientPort, socConnection);

lb\_info.Items.Add(info + "上线啦！");

lb\_info.SelectedIndex = lb\_info.Items.Count - 1;

}

catch (Exception err)

{

MessageBox.Show(err.ToString());

}

}

//获取在线用户

public void serverRecList(object socketClientPara)

{

try

{

Socket socketServer = socketClientPara as Socket;

//如果有在线用户则发送

if (dicSocket.Count != 0)

{

string info = "2\*";

foreach (var item in dicSocket)

{

info = info + item.Key + "&";

}

//向新登陆用户通知已登陆的用户清单

byte[] arrSendMsg = Encoding.UTF8.GetBytes(info);

socketServer.Send(arrSendMsg);

}

}

catch (Exception err)

{

MessageBox.Show(err.ToString());

}

}

//管理员发送消息

private void btn\_send\_Click(object sender, EventArgs e)

{

string sendMsg = "6\*" + tb\_sendInfo.Text.Trim();

byte[] arrSendMsg = Encoding.UTF8.GetBytes(sendMsg);

try

{

if (lb\_users.SelectedIndex == -1)

{

foreach (var item in dicSocket)

{

item.Value.Send(arrSendMsg);

}

}

else

{

string selectClient = lb\_users.SelectedItem.ToString();

dicSocket[selectClient].Send(arrSendMsg);

}

lb\_info.Items.Add("管理员：" + tb\_sendInfo.Text.Trim());

lb\_info.SelectedIndex = lb\_info.Items.Count - 1;

tb\_sendInfo.Text = "";

}

catch (Exception err)

{

MessageBox.Show(err.ToString());

}

}

//管理员关闭服务器

private void btn\_close\_Click(object sender, EventArgs e)

{

try

{

string sendMsg = "7\*";

byte[] arrSendMsg = Encoding.UTF8.GetBytes(sendMsg);

foreach (var item in dicSocket)

{

item.Value.Send(arrSendMsg);

}

if (socketWatch == null)

return;

setRestStatus();

isntStop = false;

socketWatch.Close();

threadWatch.Abort();

foreach (var item in dicSocket)

{

item.Value.Close();

lb\_users.Items.Clear();

}

dicSocket.Clear();

socketWatch = null;

lb\_info.Items.Add("服务器已关闭服务");

lb\_info.SelectedIndex = lb\_info.Items.Count - 1;

}

catch

{

}

}

//退出登陆

public void exitChat(object socketClientPara)

{

try

{

Socket socketServer = socketClientPara as Socket;

string info = "";

//允许退出用户的退出

byte[] arrSendMsg = Encoding.UTF8.GetBytes("8\*");

socketServer.Send(arrSendMsg);

foreach (var item in dicSocket)

{

if (item.Value == socketServer)

{

info = item.Key;

break;

}

}

dicSocket.Remove(info);

lb\_users.Items.Remove(info);

string sendMsg = "3\*" + info;

//向所有用户通知这个用户退出

arrSendMsg = Encoding.UTF8.GetBytes(sendMsg);

lb\_info.Items.Add(info + "已经退出了登录");

lb\_info.SelectedIndex = lb\_info.Items.Count - 1;

foreach (var item in dicSocket)

{

item.Value.Send(arrSendMsg);

}

}

catch (Exception err)

{

MessageBox.Show(err.ToString());

}

}

public void setRestStatus()

{

btn\_send.Enabled = false;

tb\_sendInfo.Enabled = false;

btn\_close.Enabled = false;

}

public void setWorkStatus()

{

btn\_send.Enabled = true;

tb\_sendInfo.Enabled = true;

btn\_start.Enabled = false;

btn\_close.Enabled = true;

}

private void lb\_users\_MouseClick(object sender, MouseEventArgs e)

{

var index = lb\_users.IndexFromPoint(e.X, e.Y);

lb\_users.SelectedIndex = index;

}

}

}

* 1. **客户端代码**

namespace NGchat

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

TextBox.CheckForIllegalCrossThreadCalls = false;

setRestStatus();

}

//判断聊天窗口是否有改动信息

bool isModify = false;

Socket socketClient = null;

Thread threadClient = null;

bool isntStop = true;

//点击登录按钮

private void btn\_login\_Click(object sender, EventArgs e)

{

//获取用户名

string userName = tb\_loginName.Text.Trim();

isntStop = true;

//判断用户名是否输入正确

if (string.IsNullOrEmpty(userName))

{

MessageBox.Show("请填写用户名");

return;

}

if (userName.Contains('\*'))

{

MessageBox.Show("请不要在用户名填写特殊符号");

return;

}

try

{

//建立客户端的Socket

socketClient = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);

//建立服务器的IP连接地址

IPAddress ipaddress = IPAddress.Parse("127.0.0.1");

EndPoint point = new IPEndPoint(ipaddress, 7788);

//客户端去连接服务器

socketClient.Connect(point);

//监听来自服务器的信息

threadClient = new Thread(serverRecMsg);

threadClient.IsBackground = true;

threadClient.Start();

//激活功能控件

setWorkStatus();

//给服务器发送用户名和IP信息

byte[] arrClientSendMsg = Encoding.UTF8.GetBytes("1\*" + userName);

socketClient.Send(arrClientSendMsg);

lb\_showMsg.Items.Add("【年糕聊天室】欢迎用户" + userName + "登陆聊天室");

}

catch

{

MessageBox.Show("与服务器连接失败");

}

}

//不断地监听来自服务器的消息

private void serverRecMsg()

{

try

{

while (isntStop)

{

//创建一个字节对象来储存服务器的信息

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

int length = socketClient.Receive(arrRecMsg);

string strMsg = Encoding.UTF8.GetString(arrRecMsg, 0, length);

string typeMsg = strMsg.Split('\*')[0];

string info = strMsg.Substring(2);

switch (typeMsg)

{

//其他用户上线的通知

case "1":

lb\_users.Items.Add(info);

lb\_showMsg.Items.Add("用户" + info.Split('|')[0] + "上线啦！");

lb\_showMsg.SelectedIndex = lb\_showMsg.Items.Count - 1;

break;

//获取用户在线列表

case "2":

string[] list = info.Split('&');

for (int i = 0; i < list.Length - 1; i++)

{

lb\_users.Items.Add(list[i]);

}

break;

//其他用户退出的通知

case "3":

lb\_users.Items.Remove(info);

lb\_showMsg.Items.Add(info.Split('|')[0] + "退出了登录");

lb\_showMsg.SelectedIndex = lb\_showMsg.Items.Count - 1;

break;

//收到群聊的消息

case "4":

lb\_showMsg.Items.Add(info);

lb\_showMsg.SelectedIndex = lb\_showMsg.Items.Count - 1;

break;

//收到单聊的消息

case "5":

info = strMsg.Split('\*')[1];

lb\_showMsg.Items.Add(info);

lb\_showMsg.SelectedIndex = lb\_showMsg.Items.Count - 1;

break;

//收到管理员的广播

case "6":

lb\_showMsg.Items.Add("管理员:" + info);

lb\_showMsg.SelectedIndex = lb\_showMsg.Items.Count - 1;

break;

//收到服务器被关闭的通知

case "7":

lb\_showMsg.Items.Add("聊天服务器已被关闭，谢谢您的使用。");

lb\_showMsg.SelectedIndex = lb\_showMsg.Items.Count - 1;

safeClose();

isntStop = false;

break;

//收到服务器确认用户退出的通知

case "8":

safeClose();

isntStop = false;

break;

}

}

}

catch (Exception err)

{

MessageBox.Show(err.ToString());

}

}

//点击发送信息按钮

private void btn\_send\_Click(object sender, EventArgs e)

{

try

{

if (lb\_users.SelectedIndex == -1)

{

//如果是向所有人发送消息

string sendMsg = "4\*" + tb\_loginName.Text + "\*" + tb\_inputMsg.Text.Trim();

byte[] arrClientSendMsg = Encoding.UTF8.GetBytes(sendMsg);

socketClient.Send(arrClientSendMsg);

}

else

{

//如果是指定向某个用户发送消息

string user = lb\_users.SelectedItem.ToString();

string sendMsg = "5\*" + tb\_loginName.Text + "\*" + user + "\*" + tb\_inputMsg.Text.Trim();

byte[] arrClientSendMsg = Encoding.UTF8.GetBytes(sendMsg);

socketClient.Send(arrClientSendMsg);

lb\_showMsg.Items.Add("你对 " + user.Split('|')[0] + " 说：" + tb\_inputMsg.Text.Trim());

lb\_showMsg.SelectedIndex = lb\_showMsg.Items.Count - 1;

}

tb\_inputMsg.Text = "";

}

catch (Exception err)

{

MessageBox.Show(err.ToString());

}

}

//安全关闭socket

public void safeClose()

{

try

{

if (socketClient == null)

return;

if (!socketClient.Connected)

return;

socketClient.Shutdown(SocketShutdown.Both);

socketClient.Close();

}

catch

{

}

lb\_showMsg.Items.Add("您已退出了聊天室");

lb\_showMsg.Items.Add("---------以上为历史聊天记录-----------");

lb\_users.Items.Clear();

setRestStatus();

}

//退出登陆

private void btn\_loginOut\_Click(object sender, EventArgs e)

{

byte[] arrClientSendMsg = Encoding.UTF8.GetBytes("3\*");

socketClient.Send(arrClientSendMsg);

}

//关闭窗体

private void Form1\_FormClosing(object sender, FormClosingEventArgs e)

{

try

{

if (isModify && socketClient != null)

{

if (MessageBox.Show("输入区已经改动但未发送，您还要继续关闭吗", "确认关闭", MessageBoxButtons.OKCancel, MessageBoxIcon.Question) == DialogResult.OK)

{

byte[] arrClientSendMsg = Encoding.UTF8.GetBytes("3\*");

socketClient.Send(arrClientSendMsg);

return;

}

e.Cancel = true;

}

if (socketClient != null && socketClient.Connected)

{

byte[] arrClientSendMsg = Encoding.UTF8.GetBytes("3\*");

socketClient.Send(arrClientSendMsg);

}

}

catch(Exception err)

{

MessageBox.Show(err.ToString());

}

}

//服务未开启时控件的可用性

public void setRestStatus()

{

tb\_loginName.Enabled = true;

tb\_inputMsg.Enabled = false;

btn\_send.Enabled = false;

btn\_login.Enabled = true;

btn\_loginOut.Enabled = false;

}

//服务开启时控件的可用性

public void setWorkStatus()

{

tb\_loginName.Enabled = false;

btn\_loginOut.Enabled = true;

btn\_send.Enabled = true;

btn\_login.Enabled = false;

tb\_inputMsg.Enabled = true;

}

//如果聊天窗口存在未发送的消息

private void tb\_inputMsg\_TextChanged(object sender, EventArgs e)

{

if (!string.IsNullOrEmpty(tb\_inputMsg.Text.Trim()))

{

isModify = true;

}

else

{

isModify = false;

}

}

//当点击用户列表空白处，取消选择单聊对象

private void lb\_users\_MouseClick(object sender, MouseEventArgs e)

{

var index = lb\_users.IndexFromPoint(e.X, e.Y);

lb\_users.SelectedIndex = index;

}

}

}