服务程序主要类文档

目录

[1.GPRSControlNetServer 1](#_Toc420527791)

[2.ControlCommandThread 4](#_Toc420527792)

[3.AutoCollectionThread 13](#_Toc420527793)

[4.HeartbeatThread 18](#_Toc420527794)

[5.DTU\_ClientManager 21](#_Toc420527795)

[6.modbus 26](#_Toc420527796)

[7.DTU 36](#_Toc420527797)

[8.ZG\_DTU 41](#_Toc420527798)

[9.JBT\_DTU 46](#_Toc420527799)

[10.HD\_DTU 55](#_Toc420527800)

# 1.GPRSControlNetServer

/// <summary>

/// GPRS控制器的网络服务类

/// </summary>

public class GPRSControlNetServer

{

///这个类的功能说明：首先要调用Init方法初始化socket，再调用StartListen在【Socket服务监听线程】中开启监听功能.

///然后客户端(DTU)通过网络连接socket服务，然后【Socket服务监听线程】，将这个处理客户端的任务放入本类的线程池】

///本类中的线程池的功能：等待客户端发来注册信息，并将客户端添加到客户端列表【DTU\_ClientManager】

/// <summary>

/// socket主服务

/// </summary>

static Socket ServerSocket = null;

/// <summary>

/// 监听线程

/// </summary>

static Thread RecieveThread = null;

///线程池---处理客户端第一次连接并等待注册任务

static TaskPool Pool;

/// <summary>

/// 初始化socket

/// </summary>

public static void Init()

{

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

LogManager.AddDebug("初始化socket成功");

///“处理第一次注册并等待注册”的线程池的初始化

Pool = new TaskPool();

Pool.Setmaxthread(10);

}

/// <summary>

/// 开始监听

/// </summary>

/// <param name="ipEndPoint"></param>

public static bool StartListen(string ip, string port)

{

try

{

EndPoint ipEndPoint = new IPEndPoint(IPAddress.Parse(ip), ushort.Parse(port));

ServerSocket.Bind(ipEndPoint);

ServerSocket.Listen(10);

// LogManager.AddDebug("启动服务成功");

//开启监听线程

RecieveThread = new Thread(new ThreadStart(RecieveAccept));//将接受客户端连接的方法委托给线程

RecieveThread.IsBackground = true;

RecieveThread.Start();

return true;

}

catch (Exception)

{

return false;

}

}

/// <summary>

///接收客户端的连接

/// </summary>

private static void RecieveAccept()

{

while (true)

{

try

{

Socket clientSocket = ServerSocket.Accept();

LogManager.AddDebug(String.Format("接收到连接"));

//这里交给线程池

Pool.AddTaskItem(new WaitCallback(FirstConnect), clientSocket);

// ThreadPool.QueueUserWorkItem(new WaitCallback(FirstConnect), clientSocket);

}

catch (Exception)

{

}

}

}

/// <summary>

/// 处理第一次注册并等待注册

/// </summary>

/// <param name="obj"></param>

private static void FirstConnect(object obj)

{

Socket clientSocket = (Socket)obj;

LogManager.AddDebug("连接时Socket.HashCode=" + clientSocket.GetHashCode());

string protocol = "";

bool register = false; //是否已注册

DateTime start = DateTime.Now;

DateTime now = DateTime.Now;

while (!register && start.AddMinutes(1).CompareTo(now) > 0)

{

try

{

byte[] content = new byte[GlobalPara.CONTENT\_LENGTH];

int conLen = 0;

byte type = 0;

string tel = "";

Util.DTU.rdata(ref protocol, clientSocket, GlobalPara.ReceiveTimeOut, ref content, ref conLen, ref type, ref tel); //1秒超时 等待注册数据

if (Util.DTU.ResponseRegisterOrHeart(protocol, clientSocket, tel))

{

DTU\_ClientManager.AddClient(tel, clientSocket);

register = true;

}

}

catch (SocketException)

{

}

catch (Exception ex)

{

LogManager.AddError(ex);

}

}

}

}

# 2.ControlCommandThread

/// <summary>

/// 执行control\_command表的指令

/// </summary>

public class ControlCommandThread

{

//类的主要功能说明：循环每个客户端，查找每个客户端是否有需要执行的指令，如果有，则执行指令。

/// <summary>

/// 线程池

/// </summary>

static TaskPool Pool;

/// <summary>

/// 循环客户端列表，并分配任务的线程

/// </summary>

private static Thread thread;

/// <summary>

/// 用于清空缓冲区函数中。

/// </summary>

private static byte[] buffer = new byte[1024];

/// <summary>

/// 开始这个类的任务 初始化线程池，并开始【分配任务】的线程

/// </summary>

public static void Start()

{

Pool = new TaskPool();

Pool.Setmaxthread(10);

thread = new Thread(new ThreadStart(Work));

thread.IsBackground = true;

thread.Start();

}

/// <summary>

/// 循环客户端列表，将客户端添加到线程池队列

/// </summary>

private static void Work()

{

while (true)

{

int length = DTU\_ClientManager.Clients.Count;

for (int i = 0; i < length; i++)

{

if (DTU\_ClientManager.Clients[i].StationId != 0) //如果这个客户端在数据库中有对应的站点id

Pool.AddTaskItem(new WaitCallback(ExecuteOrder), DTU\_ClientManager.Clients[i]);

}

LogManager.AddDebug("ControlCommandThread线程池 " + Pool.GetPoolDetail());

Thread.Sleep(15 \* 1000);

}

}

/// <summary>

/// 执行任务

/// </summary>

/// <param name="state"></param>

private static void ExecuteOrder(object state)

{

//1.锁住客户端对象

//2.找出客户端对应的数据库，并找出需要执行的控制指令。

//3.循环发送指令到客户端，再接收返回的数据，存入数据库。

//4.释放客户端对象

DTUClientInfo client = (DTUClientInfo)state;

using (MyLock mylock = new MyLock(client, 10000, false))

{

if (mylock.IsTimeout == false)

{

LogManager.AddDebug("ControlCommandThread线程：" + Thread.CurrentThread.ManagedThreadId + " 开始锁了 是否超时" + mylock.IsTimeout);

//找出这个站点对应的数据库, 并获取数据库连接

SqlConnection connection = GetDbConnectionByTel(client.TelOrGprsId.ToString());

if (connection == null) //找不到对应的数据库连接

{

LogManager.AddDebug(string.Format("根据Tel={0}的设备唯一id找不到对应的数据库连接", client.TelOrGprsId.ToString()));

return;

}

SWSDataContext db = new SWSDataContext(connection);

takePhoto(db, client.StationId, client);

//把那些不需要执行了的数据 state=1 就当这行指令已经执行过了

db.ExecuteCommand("UPDATE dbo.control\_command SET state=1 WHERE station\_id=" + client.StationId + " AND (state=0 OR state IS NULL) AND id NOT IN (SELECT MAX(id) FROM dbo.control\_command WHERE station\_id=" + client.StationId + " AND (state=0 OR state IS NULL) GROUP BY gong\_kuang\_id,read\_or\_write) ");

//获取需要执行的指令

List<control\_command> commands = db.ExecuteQuery<control\_command>("SELECT \* FROM dbo.control\_command WHERE id IN(SELECT MAX(id) FROM dbo.control\_command WHERE station\_id=" + client.StationId + " AND (state=0 OR state IS NULL) GROUP BY gong\_kuang\_id,read\_or\_write) ORDER BY add\_datetime").ToList();

List<gong\_kuang\_config> gongKuangs = db.gong\_kuang\_config.Where(c => c.station\_id == client.StationId).ToList();

foreach (control\_command command in commands) //循环发送指令

{

if (command.COUNT == null)

{

command.COUNT = 0;

}

try

{

gong\_kuang\_config gongkuang = gongKuangs.SingleOrDefault(c => c.id == command.gong\_kuang\_id);

if (gongkuang == null) //如果找不到command对应和gongkuangid,则把这条command标记为执行失败。

{

LogManager.AddError("找不到控制命令所对应的工况配置信息");

command.execute\_result = 0; //标记为执行失败

command.state = 1; // 就当已经执行过

command.execute\_comment = "执行失败,找不到控制命令所对应的工况配置信息";

command.complete\_datetime = DateTime.Now;

db.SubmitChanges();

continue;

}

long value = long.MinValue;

byte address = (byte)gongkuang.address;

ClearSocketCache(client.socket);//清除缓冲区

LogManager.AddDebug("hashcode=" + client.socket.GetHashCode());

if (command.read\_or\_write == 0) //读读读读读读读读读读读

{

bool flag = false;

while (flag == false && command.COUNT < SysConfig.userProfile.ExecuteFailureCount)

{

flag = modbus.readdata(client.Protocol, client.socket, "", address, ushort.Parse(gongkuang.read\_register), int.Parse(gongkuang.function\_code), gongkuang.data\_type, gongkuang.decode\_order, (int)gongkuang.receive\_timeout, ref value);

LogManager.AddDebug(String.Format("执行读命令 GPRSID:{0} gongkuangid:{1} address{2} 读寄存器编号:{3} 读取的值{4} 方法的返回bool值:{5}", client.TelOrGprsId, gongkuang.id, address, gongkuang.read\_register, value.ToString(), flag));

command.COUNT++;

}

command.state = 1; //标志为已发送

if (flag == true)

{

command.execute\_result = 1; //标记为执行成功

command.value = Convert.ToString(value);

command.execute\_comment = "执行读取操作成功";

gongkuang.read\_value = Convert.ToString(value);

gongkuang.execute\_comment = "执行读取操作成功";

if (gongkuang.config\_type == "05")

{

try

{

test test = db.test.Single(c => c.testid == gongkuang.testid);

test.value = Convert.ToString(value \* gongkuang.Multiple + gongkuang.AddNumber);

}

catch (Exception ex)

{

LogManager.AddError(ex);

LogManager.AddDebug("找不到工况对应的检测点，gongkuangid=" + gongkuang.id + " testid=" + gongkuang.testid);

}

}

}

else

{

command.execute\_result = 0; //标记为执行失败

command.execute\_comment = "执行读取操作失败";

gongkuang.execute\_comment = "执行读取操作失败";

}

}

else //写写写写写写写写写写写写写写写

{

bool flag = false;

while (flag == false && command.COUNT < SysConfig.userProfile.ExecuteFailureCount)

{

flag = modbus.writedata(client.Protocol, client.socket, "", address, ushort.Parse(gongkuang.write\_register), gongkuang.data\_type, (int)gongkuang.receive\_timeout, short.Parse(command.value));

LogManager.AddDebug(String.Format("执行写命令 GPRSID:{0} address{1} 写寄存器编号:{2} 写入的值{3} 方法的返回bool值:{4}", client.TelOrGprsId, address, gongkuang.write\_register, command.value, flag));

command.COUNT++;

}

command.state = 1; //标志为已发送

if (flag == true)

{

command.execute\_result = 1; //标记为执行成功

command.execute\_comment = "执行写操作成功";

gongkuang.write\_value = command.value;

gongkuang.execute\_comment = "执行写操作成功";

}

else

{

command.execute\_result = 0; //标记为执行失败

command.execute\_comment = "执行写操作失败";

gongkuang.execute\_comment = "执行写操作失败";

}

}

gongkuang.execute\_datetime = DateTime.Now;

command.complete\_datetime = DateTime.Now;

}

catch (Exception ex)

{

command.execute\_result = 0; //标记为执行失败

command.execute\_comment = "执行命令失败,服务程序出现了异常";

// MessageQueue.Enqueue\_DataInfo(string.Format("接收时间:【{0}】,站点：{1}, 执行工况配置失败.程序出现异常,请查看日志. ", DateTime.Now, station.name));

LogManager.AddError(ex);

}

finally

{

command.complete\_datetime = DateTime.Now;

}

db.SubmitChanges();

}

LogManager.AddDebug("ControlCommandThread线程：" + Thread.CurrentThread.ManagedThreadId + " 释放锁了 " );

}

else {

LogManager.AddDebug("ControlCommandThread 锁失败了");

}

}

}

/// <summary>

/// 拍照方法

/// </summary>

/// <param name="db">数据库对象</param>

/// <param name="stationid">站点id</param>

/// <param name="client">客户端</param>

private static void takePhoto(SWSDataContext db, int stationid, Clazz.DTUClientInfo client)

{

///根据站点id去指定的数据库中找拍照指令，如果存在拍照的指令，则调用【\_485\_JBT\_Camera】类的拍照方法。

try

{

List<control\_command> jbt\_commands = db.control\_command.Where(c => c.gong\_kuang\_id == 0 && c.state == 0 && c.tp == "3" && c.station\_id == stationid).OrderBy(c => c.add\_datetime).ToList();

foreach (control\_command command in jbt\_commands)

{

if (command.COUNT == null)

{

command.COUNT = 0;

}

try

{

bool flag = false;

command.state = 1; //标志为已发送

byte[] imageBytes = new byte[1]; //takePhoto方法内部会再次分配图像数组大小

while (flag == false && command.COUNT < SysConfig.userProfile.ExecuteFailureCount)

{

flag = \_485\_JBT\_Camera.TakePhoto(client, 5000, ref imageBytes);

command.COUNT++;

LogManager.AddDebug("拍照失败" + (int)command.COUNT + "次");

}

if (flag == true) //如果拍照成功

{

SavePicture(db, stationid, imageBytes);//保存照片到数据库

command.execute\_result = 1; //标记为执行成功

command.execute\_comment = "拍照成功";

}

else

{

command.execute\_result = 0; //标记为执行失败

command.execute\_comment = "拍照失败";

}

}

catch (Exception ex)

{

LogManager.AddError(ex);

command.execute\_result = 0;

command.execute\_comment = "拍照失败";

}

finally

{

command.complete\_datetime = DateTime.Now;

db.SubmitChanges();

}

}

}

catch (Exception ex)

{

LogManager.AddError(ex);

}

}

/// <summary>

/// 保存图片到数据库

/// </summary>

/// <param name="db"></param>

/// <param name="stationId"></param>

/// <param name="imageBytes"></param>

/// <returns></returns>

private static bool SavePicture(SWSDataContext db, int stationId, byte[] imageBytes)

{

try

{

picture\_info pic = new picture\_info();

pic.station\_id = stationId.ToString();

pic.title = "在线抓拍";

pic.contents = "在线抓拍";

pic.username = "";

pic.images = imageBytes;

pic.updatetime = DateTime.Now;

pic.source = "03";

pic.type = "01";

db.picture\_info.InsertOnSubmit(pic);

db.SubmitChanges();

return true;

}

catch (Exception ex)

{

LogManager.AddError(ex);

return false;

}

}

/// <summary>

/// 根据Tel(dtu编号,设备唯一编号)获取数据库连接

/// </summary>

/// <returns></returns>

private static SqlConnection GetDbConnectionByTel(String tel)

{

try

{

//先根据站点id找到orgid ,再根据orgid找出dbname

Clazz.Config.XML\_Station xml\_station = SysConfig.DTU\_StationConfig.Stations.SingleOrDefault(c => c.Tel == tel);

if (xml\_station != null && xml\_station.Org != null)

{

return ConnectStringHelper.GetConnection(SysConfig.userProfile.DbAddress, xml\_station.Org.DBName, SysConfig.userProfile.DbUserName, SysConfig.userProfile.DbPassword);

}

else

{

return null;

}

}

catch (Exception)

{

return null;

}

}

/// <summary>

/// 清空缓存

/// </summary>

/// <param name="socket"></param>

private static void ClearSocketCache(Socket socket)

{

try

{

socket.Receive(buffer, 1024, SocketFlags.None);

}

catch

{

}

}

}

# 3.AutoCollectionThread

/// <summary>

/// 定时采集SysConfig.xml中所有的检测点

/// </summary>

public class AutoCollectionThread

{

/// <summary>

/// 线程池

/// </summary>

static TaskPool Pool;

/// <summary>

/// 分配任务的线程

/// </summary>

private static Thread thread;

/// <summary>

/// 用于清空缓冲区函数中。

/// </summary>

private static byte[] buffer = new byte[1024];

/// <summary>

/// 开始这个类的任务 初始化线程池，并开始【分配任务】的线程

/// </summary>

public static void Start()

{

Pool = new TaskPool();

Pool.Setmaxthread(50);

thread = new Thread(new ThreadStart(Work));

thread.IsBackground = true;

thread.Start();

}

/// <summary>

/// 分配任务

/// </summary>

private static void Work()

{

while (true)

{

int length = DTU\_ClientManager.Clients.Count;

for (int i = 0; i < length; i++)

{

if (DTU\_ClientManager.Clients[i].StationId != 0)//如果这个客户端在数据库中有对应的站点id

Pool.AddTaskItem(new WaitCallback(ExecuteOrder), DTU\_ClientManager.Clients[i]);

}

LogManager.AddDebug("AutoCollectionThread线程池 " + Pool.GetPoolDetail());

Thread.Sleep(120 \* 1000);

}

}

private static void ExecuteOrder(object state)

{

///第一步：找出需要采集的指标

///第二步：循环采集的指标，取出一个指标，

///第三步：锁住客户端

///第四步：采集

///第五步：释放客户端

///第六步：回到第二步

///结束

DTUClientInfo client = (DTUClientInfo)state;

Clazz.Config.XML\_Station station = SysConfig.DTU\_StationConfig.GetStationByTel(client.TelOrGprsId);

List<Clazz.Config.XML\_Test> listTest = station.ListTest;

if (station != null && listTest != null)

{

foreach (Clazz.Config.XML\_Test test in listTest)

{

using (MyLock mylock = new MyLock(client, 3000, false)) // 在循环内锁

{

if (mylock.IsTimeout == false)

{

LogManager.AddDebug("AutoCollectionThread 线程：" + Thread.CurrentThread.ManagedThreadId + " 开始锁了 是否超时" + mylock.IsTimeout);

try

{

long value = -1;

bool result = false;

int count = 0; //执行错误的次数

while (result == false && count < SysConfig.userProfile.ExecuteFailureCount)

{

LogManager.AddDebug(string.Format("开始接收 时间:{0},站点:{1} tel:{2},testid:{3}", DateTime.Now.ToString(), client.Name, client.TelOrGprsId.ToString(), test.TestId));

result = modbus.readdata(client.Protocol, client.socket, client.TelOrGprsId, test.Address, test.RegisterNo, test.FunctionCode, test.DataType, test.DecodeOrder, test.ReceiveTimeout, ref value);

count++;

}

if (result) //接收数据成功

{

if (Between(value \* test.Multiple + test.AddNumber, test.Min, test.Max)) //如果值不在取值范围内，则不要

{

SaveToDatabase(station.Org.DBName, test.TestId, value \* test.Multiple + test.AddNumber); //保存数据

LogManager.AddDebug(string.Format("接收时间:{0}, tel:{1},value:{2},testid:{3}", DateTime.Now.ToString(), client.TelOrGprsId.ToString(), value \* test.Multiple + test.AddNumber, test.TestId));

// MessageQueue.Enqueue\_DataInfo(string.Format("接收时间:【{0}】,站点：{1}，testid:{2},值：{3}", DateTime.Now.ToString(), station.Name, test.TestId, value \* test.Multiple + test.AddNumber));

}

else

{

LogManager.AddDebug(string.Format("接收时间:【{0}】,站点：{1}，testid:{2},乘以倍率之后的值：{3} 由于值不在范围内[{4},{5}]，丢弃", DateTime.Now.ToString(), station.Name, test.TestId, value \* test.Multiple + test.AddNumber, test.Min, test.Max));

}

}

else

{

//接收数据失败

LogManager.AddDebug(string.Format("接收数据失败"));

// MessageQueue.Enqueue\_DataInfo(string.Format("接收时间:【{0}】,站点：{1}，testid:{2}, 接收数据失败", DateTime.Now.ToString(), station.Name, test.TestId));

}

}

catch (SocketException)

{

}

catch (Exception ex)

{

LogManager.AddError(ex);

DEBUG.MsgBox(ex.ToString());

}

LogManager.AddDebug("AutoCollectionThread 线程：" + Thread.CurrentThread.ManagedThreadId + " 释放锁了 ");

}

else

{

LogManager.AddDebug("AutoCollectionThread 锁失败了");

}

}

}

}

}

/// <summary>

/// 保存数据到数据库

/// </summary>

/// <param name="testid"></param>

/// <param name="value"></param>

private static void SaveToDatabase(string dbname, int testid, double value)

{

LogManager.AddDebug(string.Format("testid={0} value={1}", testid, value));

SWSDataContext db = new SWSDataContext(Util.ServerSocketHelper.GetConnection(dbname));

test test = db.test.SingleOrDefault(c => c.testid == testid); //查询出检测点

if (test != null)

{

if (test.means.Trim() == "屏幕取词" || test.means.Trim() == "自动获取")

{

realrec realrec = new realrec();

realrec.testid = testid;

realrec.value = (decimal)value;

realrec.testtime = DateTime.Now;

db.realrec.InsertOnSubmit(realrec);

db.SubmitChanges(); //提交

}

}

}

private static bool Between(double value, double min, double max)

{

if (value >= min && value <= max)

{

return true;

}

else

{

return false;

}

}

}

# 4.HeartbeatThread

/// <summary>

/// 心跳线程， 间隔一段时间去采集有没有心跳数据

/// </summary>

public class HeartbeatThread

{

static TaskPool Pool; //线程池

private static Thread thread { get; set; } //循环客户端列表的线程，也就是分配任务的线程

public static Action<string> MsgHandler; //定义一个委托，将接收到字节显示到UI主线程上。

public static void Start()

{

Pool = new TaskPool();

Pool.Setmaxthread(5);

thread = new Thread(new ThreadStart(Work));

thread.IsBackground = true;

thread.Start();

}

private static void Work()

{

//每隔一段时间，循环客户端列表，对每个客户端列表进行心跳数据的采集。

while (true)

{

int length = DTU\_ClientManager.Clients.Count;

for (int i = 0; i < length; i++)

{

Pool.AddTaskItem(new WaitCallback(Execute), DTU\_ClientManager.Clients[i]);

}

LogManager.AddDebug("ControlCommandThread线程池 " + Pool.GetPoolDetail());

Thread.Sleep(1000);

}

}

private static void Execute(object state)

{

//锁住客户端连接，接收客户端的心跳数据，

//如果没有心跳数据则跳过，

//如果有心跳数据，则将客户端更新到客户端列表。

DTUClientInfo client = (DTUClientInfo)state;

using (MyLock mylock = new MyLock(client, 3000, false))

{

if (mylock.IsTimeout == false) //判断锁是否成功，如果没有超时，则表示锁成功

{

try

{

byte[] content = new byte[GlobalPara.CONTENT\_LENGTH];

int conLen = 0;

byte type = 0;

string tel = "";

string protocol = "";

Util.DTU.rdata(ref protocol, client.socket, GlobalPara.ReceiveTimeOut, ref content, ref conLen, ref type, ref tel); //1秒超时

Util.DTU.HandlerData(protocol, client.socket, content, conLen, type, tel);

byte[] pack = new byte[1024 \* 2];

client.socket.ReceiveTimeout = 50;

int packLen = client.socket.Receive(pack, SocketFlags.None);

if (MsgHandler != null)

{

MsgHandler("收到的心跳：" + CommonUtil.byteToHexStr(pack, packLen));

}

////判断收到的数据是不是心跳 如果是心跳数据，则解析出gprsId，并更新客户端列表

//int gprsId = 0;

//bool flag = GPRS\_Protocol.UnPack\_Heartbeat(pack, packLen, ref gprsId);

//if (flag)

//{

// ClientManager.AddClient(client.socket, gprsId.ToString());

// //pack[1] = Protocol.Type\_Heartbeat\_StoC;

// //这里重新组个返回心跳包。

// BackHeartBeat(client.socket, gprsId);

//}

}

catch (Exception)

{

}

}

}

}

/// <summary>

/// 给客户端回一个心跳确认包

/// </summary>

/// <param name="socket"></param>

/// <param name="gprsId"></param>

private static void BackHeartBeat(Socket socket, int gprsId)

{

byte[] pack = new byte[100];

int packLen = 0;

byte[] b\_id = CommonUtil.IntToByteArr(gprsId);

GPRS\_Protocol.Pack(b\_id, b\_id.Length, GPRS\_Protocol.Type\_Heartbeat\_StoC, GPRS\_Protocol.Channel\_None, ref pack, ref packLen);

socket.Send(pack, packLen, SocketFlags.None);

}

}

# 5.DTU\_ClientManager

/// <summary>

/// DTU客户端连接的管理类

/// </summary>

public class DTU\_ClientManager

{

public delegate void ClientsChange();

public static List<Clazz.DTUClientInfo> Clients = new List<Clazz.DTUClientInfo>();

public static Action ClientChangeHander = null;

/// <summary>

/// 添加客户端连接 (注册客户端)

/// </summary>

/// <param name="tel">电话号码</param>

/// <param name="socket">socket实例</param>

/// <param name="ip">客户端DTU的IP</param>

/// <param name="port">客户端DTU端口</param>

public static void AddClient(string tel, Socket socket)

{

LogManager.AddDebug(string.Format("进入AddClient方法 tel:{0}", tel));

Clazz.DTUClientInfo clientInfo = Clients.SingleOrDefault(c => c.TelOrGprsId == tel);

XML\_Station station = SysConfig.DTU\_StationConfig.Stations.SingleOrDefault(c => c.Tel == tel);

lock (Clients)

{

if (clientInfo == null) //如果客户端不存在，则添加，如果存在，则修改。

{

clientInfo = new Clazz.DTUClientInfo();

clientInfo.TelOrGprsId = tel; //手机号码

clientInfo.socket = socket; //客户端连接实例

clientInfo.RegisterTime = DateTime.Now; //注册时间

clientInfo.LastVisitTime = DateTime.Now; //最后一次访问时间

if (station != null)

{

clientInfo.StationId = station.StationId; //站点id

clientInfo.Protocol = station.Protocol;

}

Clients.Add(clientInfo);

}

else

{

clientInfo.socket = socket;

clientInfo.LastVisitTime = DateTime.Now; //最后一次访问时间

}

if (ClientChangeHander != null)

{

ClientChangeHander();

}

}

LogManager.AddDebug(string.Format("退出AddClient方法 tel:{0}", tel));

}

/// <summary>

/// 更新客户端的最后一次访问时间

/// </summary>

/// <param name="clientInfo"></param>

/// <param name="datetime"></param>

public static void UpdateLastVisitTime(DTUClientInfo clientInfo, DateTime datetime)

{

clientInfo.LastVisitTime = datetime;

if (ClientChangeHander != null)

{

ClientChangeHander();

}

}

/// <summary>

/// 删除客户端注册信息

/// </summary>

/// <param name="clientInfo"></param>

public static void DeleteClient(Clazz.DTUClientInfo clientInfo)

{

lock (Clients)

{

Clients.Remove(clientInfo);

LogManager.AddDebug("删除客户端注册信息 TEL:" + clientInfo.TelOrGprsId);

// MessageQueue.Enqueue\_RegAndLogout(string.Format("【{0}】: {1}退出系统", DateTime.Now.ToString(), clientInfo.Name));

if (ClientChangeHander != null)

{

ClientChangeHander();

}

}

}

/// <summary>

/// 删除客户端注册信息

/// </summary>

/// <param name="clientInfo"></param>

public static void DeleteAllClient()

{

lock (Clients)

{

Clients.RemoveAll(c => true); //删除所有客户端连接

LogManager.AddDebug("删除所有客户端连接");

// MessageQueue.Enqueue\_RegAndLogout("所有的客户端连接都已断开");

if (ClientChangeHander != null)

{

ClientChangeHander();

}

}

}

/// <summary>

/// 根据电话号码找出站点名称

/// </summary>

/// <param name="tel"></param>

/// <returns></returns>

public static string GetStationNameByTel(string tel)

{

string stationName = string.Empty;

Clazz.Config.XML\_Station station = SysConfig.DTU\_StationConfig.GetStationByTel(tel);

if (station != null)

{

stationName = station.Name;

}

else

{

bool ExistStation = false;

foreach (XML\_Org item in SysConfig.orgConfig.Orgs)

{

try

{

SWSDataContext db = new SWSDataContext(ServerSocketHelper.GetConnection(item.DBName));

country\_station s = db.country\_station.SingleOrDefault(c => c.jiankongyitiji\_version == 2 && c.transfer\_code == tel);

if (s != null)

{

ExistStation = true;

stationName = s.name;

}

}

catch (Exception ex)

{

LogManager.AddError(ex);

}

}

if (ExistStation == false)

{

stationName = "未知的客户端";

}

}

return stationName;

}

/// <summary>

/// 根据socket 查找这个站点所使用的DTU采用的传输协议

/// </summary>

/// <param name="socket"></param>

/// <returns></returns>

public static string GetProtocolBySocket(Socket socket)

{

string protocol = "";

try

{

DTUClientInfo client = Clients.SingleOrDefault(c => c.socket == socket);

if (client != null)

{

protocol = client.Protocol;

}

}

catch (Exception ex)

{

LogManager.AddError(ex);

}

return protocol;

}

/// <summary>

/// 客户端注册

/// </summary>

/// <param name="socket"></param>

/// <param name="pack"></param>

/// <param name="length"></param>

public static void RegisterDataHandler(Socket socket, string tel)

{

LogManager.AddDebug(string.Format("进入客户端注册方法RegisterDataHandler Tel:{0}", tel));

try

{

// string stationName = DTU\_ClientManager.GetStationNameByTel(tel);

// if (string.IsNullOrEmpty(stationName))

// {

// MessageQueue.Enqueue\_RegAndLogout(string.Format("【{0}】: {1}登录系统", DateTime.Now.ToString(), "未知的客户端"));

// }

// else

// {

// MessageQueue.Enqueue\_RegAndLogout(string.Format("【{0}】: {1}登录系统", DateTime.Now.ToString(), stationName));

// }

DTU\_ClientManager.AddClient(tel, socket);

}

catch (Exception ex)

{

LogManager.AddError(ex);

}

LogManager.AddDebug(string.Format("退出客户端注册方法RegisterDataHandler Tel:{0}", tel));

}

}

# 6.Modbus

/// <summary>

/// Modbus指令的生成与解析

/// </summary>

class modbus

{

public static string modbusStatus;

#region CRC Computation

private static void GetCRC(byte[] message, ref byte[] CRC)

{

//Function expects a modbus message of any length as well as a 2 byte CRC array in which to

//return the CRC values:

ushort CRCFull = 0xFFFF;

byte CRCHigh = 0xFF, CRCLow = 0xFF;

char CRCLSB;

for (int i = 0; i < (message.Length) - 2; i++)

{

CRCFull = (ushort)(CRCFull ^ message[i]);

for (int j = 0; j < 8; j++)

{

CRCLSB = (char)(CRCFull & 0x0001);

CRCFull = (ushort)((CRCFull >> 1) & 0x7FFF);

if (CRCLSB == 1)

CRCFull = (ushort)(CRCFull ^ 0xA001);

}

}

CRC[1] = CRCHigh = (byte)((CRCFull >> 8) & 0xFF);

CRC[0] = CRCLow = (byte)(CRCFull & 0xFF);

}

#endregion

#region Build Message

private static void BuildMessage(byte address, byte type, ushort start, ushort registers, ref byte[] message)

{

//Array to receive CRC bytes:

byte[] CRC = new byte[2];

message[0] = address;

message[1] = type;

message[2] = (byte)(start >> 8);

message[3] = (byte)start;

message[4] = (byte)(registers >> 8);

message[5] = (byte)registers;

GetCRC(message, ref CRC);

message[message.Length - 2] = CRC[0];

message[message.Length - 1] = CRC[1];

}

#endregion

#region Check Response

private static bool CheckResponse(byte[] response)

{

//Perform a basic CRC check:

byte[] CRC = new byte[2];

GetCRC(response, ref CRC);

if (CRC[0] == response[response.Length - 2] && CRC[1] == response[response.Length - 1])

return true;

else

return false;

}

#endregion

#region Get Response

private static void GetResponse(string protocol, Socket socket, int receiveTimeout, ref byte[] response)

{

//There is a bug in .Net 2.0 DataReceived Event that prevents people from using this

//event as an interrupt to handle data (it doesn't fire all of the time). Therefore

//we have to use the ReadByte command for a fixed length as it's been shown to be reliable.

// for (int i = 0; i < response.Length; i++)

//{

int len = 0;

byte type = 0;

string strtel = new string('0', 11);

DTU.rdata(ref protocol, socket, receiveTimeout, ref response, ref len, ref type, ref strtel);

// response[i] = (byte)();//sp.ReadByte());

// }

}

#endregion

#region Function 16 - Write Multiple Registers

/// <summary>

/// 写寄存器

/// </summary>

/// <param name="socket"></param>

/// <param name="address"></param>

/// <param name="start"></param>

/// <param name="registers"></param>

/// <param name="values"></param>

/// <returns></returns>

public static bool SendFc16(string protocol, Socket socket, string tel, byte address, ushort start, ushort registers, string dataType, int receiveTimeout, byte channel, short[] values)

{

LogManager.AddDebug("SendFc16(Socket socket, byte " + address + ", ushort " + start + ", ushort " + registers + ", string " + dataType + ", int " + receiveTimeout + ", short " + values[0] + ")");

int dataLength = 2; //默认数据长度为两个字节

if (dataType == "2") //说明是"32位无符号二进制" 所以要用4个字节

{

dataLength = 4;

}

//Message is 1 addr + 1 fcn + 2 start + 2 reg + 1 count + 2 \* reg vals + 2 CRC

byte[] message = new byte[9 + dataLength \* registers];

//Function 16 response is fixed at 8 bytes

byte[] response = new byte[8];

try

{

//Add bytecount to message:

message[6] = (byte)(registers \* dataLength);

//Put write values into message prior to sending:

for (int i = 0; i < registers; i++)

{

for (int j = 0; j < dataLength; j++)

{

message[7 + (dataLength \* i) + j] = (byte)(values[i] >> 8 \* (dataLength - j - 1)); //将value[i]转换成指定字节长度的字节数组, (高位在前,低位在后) message下标从第七个开始

}

//message[7 + 2 \* i] = (byte)(values[i] >> 8);

//message[8 + 2 \* i] = (byte)(values[i]);

}

//Build outgoing message:

BuildMessage(address, (byte)16, start, registers, ref message);

string msg = "";

for (int i = 0; i < message.Length; i++)

{

msg += "\r\n message[" + i + "]= 值:" + message[i];

}

LogManager.AddDebug("写 message.length=" + message.Length + msg);

}

catch (Exception ex)

{

LogManager.AddError(ex);

}

//Send Modbus message to Serial Port:

try

{

DTU.SendUserData(protocol, socket, message, message.Length, tel, channel);

//sp.Write(message, 0, message.Length);

GetResponse(protocol, socket, receiveTimeout, ref response);

}

catch (Exception err)

{

modbusStatus = "Error in write event: " + err.Message;

return false;

}

//Evaluate message:

if (CheckResponse(response))

{

modbusStatus = "Write successful";

return true;

}

else

{

modbusStatus = "CRC error";

return false;

}

/\* }

else

{

modbusStatus = "Serial port not open";

return false;

}

\* \*/

}

#endregion

#region Function 3 - Read Registers

/// <summary>

/// 读寄存器或线圈

/// </summary>

/// <param name="protocol">协议( ZG\_DTU、HD\_DTU或者JBT\_DTU)</param>

/// <param name="socket">socket</param>

/// <param name="tel">DTU的Id,长得跟TEL相似，都是11位的</param>

/// <param name="address">485总线上的设备地址</param>

/// <param name="RegisterStartAdd">寄存器起始地址 寄存器个数为1，方法内部已写死。</param>

/// <param name="ModbusfunctionCode">功能码 </param>

/// <param name="dataType">寄存器类型 1代表16位无符号二进制的寄存器，2代表32位的寄存器</param>

/// <param name="DecodeOrder">获取的数据的高低位顺序</param>

/// <param name="receiveTimeout">超时时间 </param>

/// <param name="channel">GPRS控制器的通道号，也是串口号</param>

/// <param name="values">返回的value值 虽然是个数组，但数据只有一个，下标为0</param>

/// <returns></returns>

public static bool SendFc(string protocol, Socket socket, string tel, byte address, ushort RegisterStartAdd,

int ModbusfunctionCode, string dataType, string DecodeOrder, int receiveTimeout, byte channel, ref long[] values)

{

int dataLength = 2; //16位无符号二进制的寄存器数据长度为两个字节 默认为16位寄存器

if (dataType == "2") //说明是"32位无符号二进制" 所以要用4个字节

{

dataLength = 4;

}

LogManager.AddDebug("dataLength=" + dataLength);

byte[] message = new byte[8];

//Function 3 response buffer:

byte[] response = new byte[5 + dataLength \* 1];

//Build outgoing modbus message:

LogManager.AddDebug(" BuildMessage(" + address + ", " + ModbusfunctionCode + ", " + RegisterStartAdd + ", 1, ref message);");

BuildMessage(address, (byte)ModbusfunctionCode, RegisterStartAdd, ushort.Parse(dataType), ref message);

//Send modbus message to Serial Port:

string msg = "";

try

{

DTU.SendUserData(protocol, socket, message, message.Length, tel, channel);

//sp.Write(message, 0, message.Length);

GetResponse(protocol, socket, receiveTimeout, ref response);

//日志

for (int i = 0; i < response.Length; i++)

{

msg += response[i] + " ";

}

LogManager.AddDebug("functionCode=" + ModbusfunctionCode + " response.length=" + response.Length + " response=" + msg);

}

catch (SocketException ex)

{

modbusStatus = ex.Message;

return false;

}

catch (Exception ex)

{

LogManager.AddError(ex);

modbusStatus = "Error in read event: " + ex.Message;

return false;

}

//处理返回结果

if (CheckResponse(response))

{

int datalength = response[2]; //数据长度(单位:字节)

LogManager.AddDebug("返回的数据长度datalength=" + datalength);

//设置默认解码顺序

if (string.IsNullOrEmpty(DecodeOrder))

{

if (datalength == 2)

{

DecodeOrder = "12";

}

else if (datalength == 4)

{//三四个字节为高位，一二两个字节为低位

DecodeOrder = "3412";

}

}

//解析返回的modbus数据

values[0] = 0;

try

{

for (int i = 0; i < DecodeOrder.Length; i++)

{

int n = Convert.ToInt32(DecodeOrder[i].ToString());

values[0] <<= 8;

values[0] += response[2 + n];

}

}

catch (Exception)

{

LogManager.AddError("解析返回的modbus时出错，modbus指令为：" + msg);

return false;

}

modbusStatus = "Read successful";

return true;

}

else

{

modbusStatus = "CRC error";

return false;

}

}

#endregion

#region Function 3 - Read Registers

/// <summary>

/// 读数据

/// </summary>

/// <param name="protocol">协议( ZG\_DTU、HD\_DTU或者JBT\_DTU)</param>

/// <param name="socket">socket</param>

/// <param name="tel">DTU的Id,长得跟TEL相似，都是11位的</param>

/// <param name="address">485总线上的设备地址</param>

/// <param name="RegisterStartAdd">寄存器起始地址 寄存器个数为1，方法内部已写死。</param>

/// <param name="functionCode">功能码</param>

/// <param name="dataType">寄存器类型 1代表16位无符号二进制的寄存器，2代表32位的寄存器</param>

/// <param name="DecodeOrder">获取的数据的高低位顺序</param>

/// <param name="receiveTimeout">超时时间</param>

/// <param name="value">返回的value值 </param>

/// <returns></returns>

public static bool readdata(string protocol, Socket socket, string tel, byte address, ushort RegisterStartAdd,

int functionCode, string dataType, string DecodeOrder, int receiveTimeout, ref long value)

{

return readdata(protocol, socket, tel, address, RegisterStartAdd, functionCode, dataType, DecodeOrder, receiveTimeout, new byte(), ref value);

}

/// <summary>

/// 读数据 （这个方法给浙江大学城市学院江老师开发的GPRS控制器调用，这个方法多了channel ）

/// </summary>

/// <param name="protocol">协议( ZG\_DTU、HD\_DTU或者JBT\_DTU)</param>

/// <param name="socket">socket</param>

/// <param name="tel">DTU的Id,长得跟TEL相似，都是11位的</param>

/// <param name="address">485总线上的设备地址</param>

/// <param name="RegisterStartAdd">寄存器起始地址 寄存器个数为1，方法内部已写死。</param>

/// <param name="functionCode">功能码</param>

/// <param name="dataType">寄存器类型 1代表16位无符号二进制的寄存器，2代表32位的寄存器</param>

/// <param name="DecodeOrder">获取的数据的高低位顺序</param>

/// <param name="receiveTimeout">超时时间</param>

/// <param name="channel">GPRS控制器的通道号，也是串口号</param>

/// <returns></returns>

public static bool readdata(string protocol, Socket socket, string tel, byte address, ushort RegisterStartAdd,

int functionCode, string dataType, string DecodeOrder, int receiveTimeout, byte channel, ref long value)

{

long[] values = new long[2];

bool result;

result = SendFc(protocol, socket, tel, address, RegisterStartAdd, functionCode, dataType, DecodeOrder, receiveTimeout, channel, ref values);

value = values[0];

return result;

}

/// <summary>

/// 写数据

/// </summary>

/// <param name="protocol">协议( ZG\_DTU、HD\_DTU或者JBT\_DTU)</param>

/// <param name="socket">socket</param>

/// <param name="tel">DTU的Id,长得跟电话号码相似，都是11位的</param>

/// <param name="address">485总线上的设备地址</param>

/// <param name="RegisterStartAdd">寄存器地址</param>

/// <param name="dataType">数据类型(1代表16位无符号二进制,2代表32位无符号二进制)</param>

/// <param name="receiveTimeout">接收超时时间(毫秒)</param>

/// <param name="value">写入的值</param>

/// <returns></returns>

public static bool writedata(string protocol, Socket socket, string tel, byte address,

ushort RegisterStartAdd, string dataType, int receiveTimeout, short value)

{

return writedata(protocol, socket, tel, address, RegisterStartAdd, dataType, receiveTimeout, new byte(), value);

}

/// <summary>

/// 写数据

/// </summary>

/// <param name="protocol">协议( ZG\_DTU、HD\_DTU或者JBT\_DTU)</param>

/// <param name="socket">socket</param>

/// <param name="tel">DTU的Id,长得跟电话号码相似，都是11位的</param>

/// <param name="address">485总线上的设备地址</param>

/// <param name="RegisterStartAdd">寄存器地址</param>

/// <param name="dataType">数据类型(1代表16位无符号二进制,2代表32位无符号二进制)</param>

/// <param name="receiveTimeout">接收超时时间(毫秒)</param>

/// <param name="channel">GPRS控制器的通道号，也是串口号</param>

/// <param name="value">写入的值</param>

/// <returns></returns>

public static bool writedata(string protocol, Socket socket, string tel, byte address,

ushort RegisterStartAdd, string dataType, int receiveTimeout, byte channel, short value)

{

short[] values = new short[2];

bool result;

values[0] = value;

result = SendFc16(protocol, socket, tel, address, RegisterStartAdd, 1, dataType, receiveTimeout, channel, values);

return result;

}

#endregion

}

# 7.DTU

/// <summary>

/// 通讯设备类型， 通讯区分具体的通讯设备调用具体的通讯协议类, ZG\_DTU或HD\_DTU或者其他的。

/// </summary>

public class DTU

{

/// <summary>

/// 佐格DTU

/// </summary>

public static readonly string \_ZG\_DTU = "ZG\_DTU";

/// <summary>

/// 宏电DTU

/// </summary>

public static readonly string \_HD\_DTU = "HD\_DTU";

/// <summary>

/// 金博通DTU

/// </summary>

public static readonly string \_JBT\_DTU = "JBT\_DTU";

/// <summary>

/// GPRS控制器 浙江大学城市学院江老师开发的版本。

/// </summary>

public static readonly string \_GPRS\_CONTROL = "GPRS\_CONTROL";

/// <summary>

/// 解析注册的数据是DTU协议还是DDP协议

/// </summary>

/// <param name="buffer"></param>

/// <returns></returns>

public static string CheckProtocolByRegisterData(byte[] buffer, int len)

{

if (ZG\_DTU.CheckProtocolByRegisterData(buffer, len) == true)

{

return \_ZG\_DTU;

}

if (HD\_DTU.CheckProtocolByRegisterData(buffer, len) == true)

{

return \_HD\_DTU;

}

if (JBT\_DTU.CheckProtocolByRegisterData(buffer, len) == true)

{

return \_JBT\_DTU;

}

//如果不符合上面的协议标准 则返回空字符串

LogManager.AddDebug("检验注册数据格式不正确 ,即不符合佐格DTU协议,也不符合宏电DTU协议,也不符号金博通DTU协议");

return "";

}

/// <summary>

/// 接收数据

/// </summary>

/// <param name="socket"></param>

/// <param name="timeout"></param>

/// <param name="content"></param>

/// <param name="conLen"></param>

/// <param name="type"></param>

/// <param name="tel"></param>

/// <returns></returns>

public static bool rdata(ref string protocol, Socket socket, int timeout, ref byte[] content, ref int conLen, ref byte type, ref string tel)

{

if (protocol == DTU.\_ZG\_DTU)

{

return ZG\_DTU.rdata(socket, timeout, ref content, ref conLen, ref type, ref tel); //1秒超时 等待注册数据

}

else if (protocol == DTU.\_HD\_DTU)

{

return HD\_DTU.rdata(socket, timeout, ref content, ref conLen, ref type, ref tel); //1秒超时 等待注册数据

}

else if (protocol == DTU.\_JBT\_DTU)

{

return JBT\_DTU.rdata(socket, timeout, ref content, ref conLen, ref type, ref tel); //1秒超时 等待注册数据

}

else

{

//在DTU未注册之前, socket接收都是调用以下的代码 ,

try

{

byte[] buffer = new byte[GlobalPara.SEND\_DATA\_LENGTH]; //接收数据的数组

socket.ReceiveTimeout = timeout;

int len = socket.Receive(buffer, buffer.Length, SocketFlags.None);

protocol = DTU.CheckProtocolByRegisterData(buffer, len);

if (protocol == DTU.\_ZG\_DTU)

{

return ZG\_DTU.UnPack(buffer, len, ref content, ref conLen, ref type, ref tel);

}

else if (protocol == DTU.\_HD\_DTU)

{

return HD\_DTU.UnPack(buffer, len, ref content, ref conLen, ref type, ref tel);

}

else if (protocol == DTU.\_JBT\_DTU)

{

return JBT\_DTU.UnPack(buffer, len, ref content, ref conLen, ref type, ref tel);

}

else

{

return false;

}

}

catch (SocketException)

{

return false;

}

catch (Exception ex)

{

LogManager.AddError(ex);

return false;

}

}

}

/// <summary>

/// 处理数据

/// </summary>

/// <param name="protocol"></param>

/// <param name="clientSocket"></param>

/// <param name="content"></param>

/// <param name="conLen"></param>

/// <param name="type"></param>

/// <param name="tel"></param>

public static bool HandlerData(string protocol, Socket clientSocket, byte[] content, int conLen, byte type, string tel)

{

if (protocol == DTU.\_ZG\_DTU)

{

ZG\_DTU.HandlerData(clientSocket, content, conLen, type, tel);

return true;

}

else if (protocol == DTU.\_HD\_DTU)

{

HD\_DTU.HandlerData(clientSocket, content, conLen, type, tel);

return true;

}

else if (protocol == DTU.\_JBT\_DTU)

{

JBT\_DTU.HandlerData(clientSocket, content, conLen, type, tel);

return true;

}

else

{

return false;

}

}

/// <summary>

/// 发送数据

/// </summary>

/// <param name="socket"></param>

/// <param name="content"></param>

/// <param name="conLen"></param>

/// <param name="type"></param>

public static void SendUserData(string protocol, Socket socket, byte[] content, int conLen, string tel, byte channel)

{

if (protocol == DTU.\_ZG\_DTU)

{

ZG\_DTU.Send(socket, content, conLen, ZG\_DTU.USER\_DATA);

}

else if (protocol == DTU.\_HD\_DTU)

{

HD\_DTU.Send(socket, content, conLen, HD\_DTU.DSC\_User\_Data, tel);

}

else if (protocol == DTU.\_JBT\_DTU)

{

JBT\_DTU.SendUserData\_Master(socket, content, conLen, tel);

}

else if (protocol == DTU.\_GPRS\_CONTROL)

{

GPRS\_Protocol.Send(socket, channel, content, conLen);

}

}

/// <summary>

/// 响应注册数据或心跳数据

/// </summary>

/// <param name="protocol"></param>

/// <param name="socket"></param>

/// <param name="tel"></param>

/// <returns></returns>

public static bool ResponseRegisterOrHeart(string protocol, Socket socket, string tel)

{

if (protocol == DTU.\_ZG\_DTU)

{

return ZG\_DTU.HeartbeatDataHandler(socket);

}

else if (protocol == DTU.\_HD\_DTU)

{

return HD\_DTU.ResponseRegister(socket, tel);

}

else if (protocol == DTU.\_JBT\_DTU)

{

return JBT\_DTU.Response\_Reigster(socket, tel);

}

else

{

return false;

}

}

}

# 8.ZG\_DTU

/// <summary>

/// 佐格DTU

/// </summary>

public class ZG\_DTU

{

/// <summary>

/// 心跳数据

/// </summary>

public static readonly byte HEARTBEAT\_DATA = 0x30;

/// <summary>

/// 注册数据

/// </summary>

public static readonly byte REGISTER\_DATA = 0x31;

/// <summary>

/// 用户数据

/// </summary>

public static readonly byte USER\_DATA = 0x32;

/// <summary>

/// 协议头

/// </summary>

public static readonly byte HEAD = 0x7b;

/// <summary>

/// 协议尾

/// </summary>

public static readonly byte END = 0x7d;

/// <summary>

/// 打包

/// </summary>

/// <param name="content">需要打包的内容</param>

/// <param name="conLen">需要打包的内容的长度</param>

/// <param name="pack">打包后的数据</param>

/// <param name="type">类型</param>

/// <param name="packLen">打包后的数据的长度</param>

/// <param name="DTU\_Id">DTU\_Id 通常为11位电话号码</param>

/// <returns></returns>

public static void Pack(byte[] content, int conLen, byte type, ref byte[] pack, ref int packLen)

{

pack[0] = HEAD; //头

pack[1] = type; //类型

byte[] sss = BitConverter.GetBytes(conLen);

pack[2] = sss[0];

pack[3] = sss[1];

Buffer.BlockCopy(content, 0, pack, 4, conLen); //内容

pack[4 + +conLen] = END;

packLen = 4 + +conLen + 1;

}

/// <summary>

/// 拆包

/// </summary>

/// <param name="str"></param>

/// <param name="strcon"></param>

/// <param name="type"></param>

/// <param name="strlen"></param>

/// <param name="conlen"></param>

/// <returns></returns>

public static bool UnPack(byte[] pack, int packLen, ref byte[] content, ref int conlen, ref byte type, ref string tel)

{

try

{

//日志

string str = "";

for (int j = 0; j < packLen; j++)

{

str += pack[j] + " ";

}

LogManager.AddDebug(str);

int i;

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

{

if (pack[i] == ZG\_DTU.HEAD)

{

break;

}

}

type = pack[i + 1]; //类型

if (type == ZG\_DTU.USER\_DATA) //用户数据拆包

{

int low = pack[i + 2]; //低8位

int high = pack[i + 3]; //高8位

int length = high \* 256 + low; //从手机号码到最后的字节数

conlen = length; //内容长度

Buffer.BlockCopy(pack, i + 4, content, 0, conlen);

}

else if (type == ZG\_DTU.REGISTER\_DATA) //注册数据拆包

{

int low = pack[i + 2]; //低8位

int high = pack[i + 3]; //高8位

int length = high \* 256 + low; //从手机号码到最后的字节数

byte[] byteTel = new byte[11]; //手机号码字节

Buffer.BlockCopy(pack, i + 4, byteTel, 0, 11);

tel = Encoding.ASCII.GetString(byteTel);

conlen = length - 11; //内容长度

Buffer.BlockCopy(pack, i + 4 + 11, content, 0, conlen);

}

}

catch (Exception ex)

{

LogManager.AddError(ex);

return false;

}

return true;

}

/// <summary>

/// 发送数据

/// </summary>

/// <param name="socket"></param>

/// <param name="content"></param>

/// <param name="conLen"></param>

/// <param name="type"></param>

public static void Send(Socket socket, byte[] content, int conLen, byte type)

{

byte[] pack = new byte[GlobalPara.SEND\_DATA\_LENGTH];

int packLen = 0;

Pack(content, conLen, type, ref pack, ref packLen);

socket.SendTimeout = GlobalPara.SendTimeOut; //设置超时

socket.Send(pack, packLen, SocketFlags.None);

}

/// <summary>

/// 接收数据

/// </summary>

/// <param name="socket"></param>

/// <param name="timeout"></param>

/// <param name="content"></param>

/// <param name="conlen"></param>

/// <param name="type"></param>

/// <param name="tel"></param>

/// <returns></returns>

public static bool rdata(Socket socket, int timeout, ref byte[] content, ref int conlen, ref byte type, ref string tel)

{

byte[] buffer = new byte[GlobalPara.SEND\_DATA\_LENGTH]; //接收数据的数组

socket.ReceiveTimeout = timeout;

int len = socket.Receive(buffer, buffer.Length, SocketFlags.None);

return UnPack(buffer, len, ref content, ref conlen, ref type, ref tel);

}

/// <summary>

/// 处理接收到的数据

/// </summary>

/// <param name="asyncResult"></param>

public static void HandlerData(Socket socket, byte[] content, int conLen, byte type, string tel)

{

try

{

if (type == ZG\_DTU.HEARTBEAT\_DATA) //心跳数据

{

HeartbeatDataHandler(socket);

}

else if (type == ZG\_DTU.REGISTER\_DATA) //注册数据 貌似这里好像不会出现注册数据了,因为注册数据在第一次注册后就不会再注册了

{

DTU\_ClientManager.AddClient(tel, socket);

}

else if (type == ZG\_DTU.USER\_DATA)

{

}

}

catch (Exception ex)

{

//LogManager.Add("json字符串:" + data + System.Environment.NewLine + ex.ToString());

LogManager.AddError(ex);

DEBUG.ThrowException(ex);

}

}

/// <summary>

/// 心跳数据处理

/// </summary>

/// <param name="socket"></param>

/// <param name="pack"></param>

public static bool HeartbeatDataHandler(Socket socket)

{

try

{

//对心跳做一个回应

ZG\_DTU.Send(socket, new byte[1], 0, ZG\_DTU.HEARTBEAT\_DATA);

return true;

}

catch (Exception ex)

{

LogManager.AddError(ex);

DEBUG.MsgBox(ex.ToString());

return false;

}

}

public static bool CheckProtocolByRegisterData(byte[] pack, int len)

{

bool flag = true;

try

{

int i;

for (i = 0; i < len; i++)

{

if (pack[i] == ZG\_DTU.HEAD) //先找到头

{

break;

}

}

byte type = pack[i + 1];

if (type != ZG\_DTU.REGISTER\_DATA) //再判断时否是ZG\_DTU的注册类型

flag = false;

int low = pack[i + 2]; //低8位

int high = pack[i + 3]; //高8位

int length = high \* 256 + low; //从手机号码到最后的字节数

if (pack[i + 1 + 2 + length + 1] != END) //再找到尾

{

flag = false;

}

}

catch (Exception ex)

{

LogManager.AddError(ex);

flag = false;

}

return flag;

}

}

# 9.JBT\_DTU

/// <summary>

/// 金博通DTU

/// </summary>

public class JBT\_DTU

{

///摄像头是备用串口，另一个是主串口

///// <summary>

///// 主用串口

///// </summary>

//public readonly int Master = 0;

///// <summary>

///// 备用串口

///// </summary>

//public readonly int Master = "Master";

#region function\_code 功能码

/// <summary>

/// DTU主串口向服务器上传数据

/// </summary>

public static readonly byte DTU\_to\_Server\_SendData\_Master = 0x01;

/// <summary>

/// DTU向服务器发送注册或心跳数据

/// </summary>

public static readonly byte DTU\_to\_Server\_Register = 0x02;

/// <summary>

/// 服务器向DTU响应注册或心跳数据

/// </summary>

public static readonly byte Server\_to\_DTU\_Register = 0x03;

/// <summary>

/// DTU备用串口向服务器上传数据

/// </summary>

public static readonly byte DTU\_to\_Server\_SendData\_Assistant = 0x11;

/// <summary>

/// 服务器向DTU备用串口下发数据

/// </summary>

public static readonly byte Server\_to\_DTU\_SendData\_Assistant = 0x12;

#endregion

/// <summary>

/// 帧头0

/// </summary>

public static readonly byte Head0 = 0xA8;

/// <summary>

/// 帧头1

/// </summary>

public static readonly byte Head1 = 0x81;

public static readonly int ReceiveLength = 4096; //理论上一个包只有1028，但考虑到可能会发生多个包在一起的现象，所以接收长度就给得大了点

/// <summary>

/// 打包备用串口用户数据 也就是摄像头的数据

/// </summary>

/// <param name="content">内容</param>

/// <param name="contentLen">内容长度</param>

/// <param name="SerialNum">主用串口还是备用串口，这两个串口的协议一点区别</param>

private static bool PackUserData\_Master(byte[] content, int contentLen, string tel, ref byte[] pack, ref int packLen)

{

pack[0] = Head0;

pack[1] = Head1;

if (tel.Length != 11) //手机号必须是11位的

{

return false;

}

byte[] b\_tel = Encoding.ASCII.GetBytes(tel);

Buffer.BlockCopy(b\_tel, 0, pack, 2, 11);

pack[13] = 0x30;//区号

pack[14] = 0x30;//区号

pack[15] = 0x30;//区号

pack[16] = 0x30;//区号

pack[17] = Server\_to\_DTU\_SendData\_Assistant; //功能码

pack[18] = 0x00; //在配置DTU的时候区号一律不启用

GenerateCheckCode(pack);

Buffer.BlockCopy(content, 0, pack, 20, contentLen);

packLen = 20 + contentLen;

return true;

}

public static bool UnPack(byte[] pack, int packLen, ref byte[] content, ref int conLen, ref byte type, ref string tel)

{

try

{

int i;

for (i = 0; i < packLen; i++)

{

if (pack[i] == JBT\_DTU.Head0 && pack[i + 1] == JBT\_DTU.Head1)

{

break;

}

}

tel = Encoding.ASCII.GetString(pack, i + 2, 11);

//手机号后四位是区号， 没什么用，就不读取了

type = pack[i + 2 + 11 + 4];

Buffer.BlockCopy(pack, 20, content, 0, packLen - 20);

conLen = packLen - 20;

//处理多个包的情况 ，暂时不处理 ，有空了再写

//int contentIndex;

//for (contentIndex = 20; i < packLen - 1; i++)

//{

// if (pack[i] == JBT\_DTU.Head0 && pack[i + 1] == JBT\_DTU.Head1)

// {

// if (i+2)

// break;

// }

//}

//if (i == (packLen - 1)) //pack中只有一个包

//{

// Buffer.BlockCopy(pack, 20, content, 0, packLen - 20);

// conLen = packLen - 20;

//}

//else //当出现有多个包连在一起的时候会执行下面的语句

//{

// Buffer.BlockCopy(pack, 20, content, 0, i - 20);

// conLen = i - 20;

//}

return true;

}

catch (Exception ex)

{

//LogManager.AddError(ex);

return false;

}

}

public static void HandlerData(Socket clientSocket, byte[] content, int conLen, byte type, string tel)

{

if (type == HD\_DTU.DTU\_REGISTER\_DATA)

{

if (Response\_Reigster(clientSocket, tel))

{

DTU\_ClientManager.AddClient(tel, clientSocket);

}

else

{

LogManager.AddDebug("tel=" + tel + " 响应客户端注册数据失败");

}

}

}

/// <summary>

/// 回应注册

/// </summary>

/// <param name="ReceiveBuffer">接收到的注册数据或心跳数据</param>

/// <param name="len">接收到的字节数</param>

public static bool Response\_Reigster(Socket socket, string tel)

{

try

{

byte[] sendBytes = new byte[20];

//两字节的头

sendBytes[0] = 0xA8;

sendBytes[1] = 0x81;

//11个字节的手机号

if (tel == null && tel.Length != 11)

{

return false;

}

Buffer.BlockCopy(Encoding.ASCII.GetBytes(tel), 0, sendBytes, 2, 11);

//四个字节的区号

sendBytes[13] = 0x30;

sendBytes[14] = 0x30;

sendBytes[15] = 0x30;

sendBytes[16] = 0x30;

//一个字节的功能码

sendBytes[17] = JBT\_DTU.Server\_to\_DTU\_Register;

//一个字节的区号启用标志

sendBytes[18] = 0x00;

//生成校验码

GenerateCheckCode(sendBytes);

int n = socket.Send(sendBytes, sendBytes.Length, SocketFlags.None);

return n > 0;

}

catch (Exception ex)

{

LogManager.AddError(ex);

return false;

}

}

/// <summary>

/// 发送主用串口数据

/// </summary>

/// <param name="socket">套接字</param>

/// <param name="content">内容</param>

/// <param name="contentLen">内容长度 </param>

/// <param name="tel">手机号</param>

/// <returns></returns>

public static int SendUserData\_Master(Socket socket, byte[] content, int contentLen, string tel)

{

byte[] pack = new byte[GlobalPara.SEND\_DATA\_LENGTH];

int packLen = 0;

Buffer.BlockCopy(content, 0, pack, 0, contentLen);

packLen = contentLen;

// bool flag = Pack(content, contentLen, tel, function\_code, SerialNum, ref pack, ref packLen);

//if (!flag) { return 0; }

//这里的内容是透传的，所以content外面没有包任何协议。

int n = Send(socket, pack, packLen);

return n;

}

/// <summary>

/// 发送备用串口数据 也就是摄像头的数据

/// </summary>

/// <param name="socket"></param>

/// <param name="content"></param>

/// <param name="contentLen"></param>

/// <param name="tel"></param>

/// <returns></returns>

public static int SendUserData\_Assistant(Socket socket, byte[] content, int contentLen, string tel)

{

byte[] pack = new byte[GlobalPara.SEND\_DATA\_LENGTH];

int packLen = 0;

bool flag = PackUserData\_Master(content, contentLen, tel, ref pack, ref packLen);

if (!flag) { return 0; }

int n = Send(socket, pack, packLen);

return n;

}

/// <summary>

/// 发送数据

/// </summary>

/// <param name="socket"></param>

/// <param name="pack"></param>

/// <param name="packLen"></param>

/// <returns></returns>

public static int Send(Socket socket, byte[] pack, int packLen)

{

socket.SendTimeout = GlobalPara.SendTimeOut; //设置超时

return socket.Send(pack, packLen, SocketFlags.None);

}

///// <summary>

///// 当收到注册信息无误后, 应调用此方法回应DTU

///// </summary>

//public static void ResponseRegister(Socket socket, string tel)

//{

// //应答客户端

// byte[] sendBytes = new byte[16];

// sendBytes[0] = HD\_DTU.HEAD;

// sendBytes[1] = HD\_DTU.DSC\_REGISTER\_DATA;

// sendBytes[2] = 0;

// sendBytes[3] = 0x10;

// byte[] b\_tel = Encoding.ASCII.GetBytes(tel);

// Buffer.BlockCopy(b\_tel, 0, sendBytes, 4, 11);

// sendBytes[4 + 11] = HD\_DTU.END;

// int n = Send(socket, sendBytes, 16);

//}

/// <summary>

/// 生成校验码

/// </summary>

/// <param name="command"></param>

private static void GenerateCheckCode(byte[] command)

{

int CheckCode = 0;

for (int i = 0; i < 19; i++)

{

CheckCode += command[i];

}

command[19] = (byte)(CheckCode % 256); //低位

}

/// <summary>

/// 接收数据

/// </summary>

/// <param name="socket">套按字</param>

/// <param name="timeout">接收超时（毫秒）</param>

/// <param name="content">接收的内容</param>

/// <param name="conLen">内容长度</param>

/// <param name="type">功能码</param>

/// <param name="tel">手机号</param>

/// <returns></returns>

public static bool rdata(Socket socket, int timeout, ref byte[] content, ref int conLen, ref byte type, ref string tel)

{

byte[] buffer = new byte[JBT\_DTU.ReceiveLength]; //接收数据的数组

socket.ReceiveTimeout = timeout;

int packLen = socket.Receive(buffer, buffer.Length, SocketFlags.None);

return UnPack(buffer, packLen, ref content, ref conLen, ref type, ref tel);

}

public static bool CheckProtocolByRegisterData(byte[] pack, int len)

{

try

{

int i;

for (i = 0; i < len; i++)

{

if (pack[i] == JBT\_DTU.Head0 && pack[i + 1] == JBT\_DTU.Head1) //先找到头

{

break;

}

}

if (i == len)

{

return false;

}

if (pack[i + 17] != DTU\_to\_Server\_Register)

{

return false;

}

return CheckCheckCode(pack.Skip(i).Take(20).ToArray());

}

catch (Exception ex)

{

LogManager.AddError(ex);

return false;

}

}

/// <summary>

/// 检验校验码是否正确

/// </summary>

/// <param name="pack"></param>

/// <returns></returns>

private static bool CheckCheckCode(byte[] pack)

{

int CheckCode = 0;

for (int i = 0; i < 19; i++)

{

CheckCode += pack[i];

}

return pack[19] == (byte)(CheckCode % 256);

}

}

# 10.HD\_DTU

/// <summary>

/// 宏电DTU

/// </summary>

public class HD\_DTU

{

/// <summary>

/// 协议头

/// </summary>

private static readonly byte HEAD = 0x7b;

/// <summary>

/// 协议尾

/// </summary>

private static readonly byte END = 0x7b;

private static int ReceiveLength = 1024;

#region DSC端协议包类型

public static readonly byte DSC\_REGISTER\_DATA = 0x81;

public static readonly byte DSC\_User\_Data = 0x89;

#endregion

#region DTU端协议包类型

public static readonly byte DTU\_REGISTER\_DATA = 0x01;

public static readonly byte DTU\_User\_Data = 0x09;

#endregion

/// <summary>

/// 接收数据

/// </summary>

/// <param name="socket"></param>

/// <param name="timeout"></param>

/// <param name="content"></param>

/// <param name="conlen"></param>

/// <param name="type"></param>

/// <param name="tel"></param>

/// <returns></returns>

public static bool rdata(Socket socket, int timeout, ref byte[] content, ref int conlen, ref byte type, ref string tel)

{

byte[] buffer = new byte[HD\_DTU.ReceiveLength]; //接收数据的数组

socket.ReceiveTimeout = timeout;

int packLen = socket.Receive(buffer, buffer.Length, SocketFlags.None);

return UnPack(buffer, packLen, ref content, ref conlen, ref type, ref tel);

}

/// <summary>

/// 发送数据

/// </summary>

/// <param name="socket"></param>

/// <param name="pack"></param>

/// <param name="packLen"></param>

/// <returns></returns>

public static int Send(Socket socket, byte[] pack, int packLen)

{

socket.SendTimeout = GlobalPara.SendTimeOut; //设置超时

return socket.Send(pack, packLen, SocketFlags.None);

}

/// <summary>

/// 发送数据

/// </summary>

/// <param name="socket"></param>

/// <param name="content"></param>

/// <param name="conLen"></param>

/// <param name="type"></param>

public static int Send(Socket socket, byte[] content, int conLen, byte type, string tel)

{

byte[] pack = new byte[GlobalPara.SEND\_DATA\_LENGTH];

int packLen = 0;

if (type == DSC\_User\_Data)

{

PackUserData(content, conLen, tel, ref pack, ref packLen);

}

int n = Send(socket, pack, packLen);

return n;

}

public static void HandlerData(Socket clientSocket, byte[] content, int conLen, byte type, string tel)

{

if (type == HD\_DTU.DTU\_REGISTER\_DATA)

{

DTU\_ClientManager.AddClient(tel, clientSocket);

ResponseRegister(clientSocket, tel);

}

}

/// <summary>

/// 打包

/// </summary>

/// <param name="content">需要打包的内容</param>

/// <param name="conLen">需要打包的内容的长度</param>

/// <param name="pack">打包后的数据</param>

/// <param name="type">类型</param>

/// <param name="packLen">打包后的数据的长度</param>

/// <param name="DTU\_Id">DTU\_Id 通常为11位电话号码</param>

/// <returns></returns>

public static void PackUserData(byte[] content, int conLen, string tel, ref byte[] pack, ref int packLen)

{

pack[0] = HEAD; //头

pack[1] = DSC\_User\_Data; //类型

packLen = conLen + 1 + 1 + 2 + 11 + 1;

byte[] sss = BitConverter.GetBytes(packLen);

pack[2] = sss[1]; //高位在前

pack[3] = sss[0]; //低位在后

//tel

byte[] b\_tel = new byte[11];

b\_tel = Encoding.ASCII.GetBytes(tel);

Buffer.BlockCopy(b\_tel, 0, pack, 4, 11);

Buffer.BlockCopy(content, 0, pack, 15, conLen); //内容

pack[15 + conLen] = END; //协议尾

}

/// <summary>

/// 检查注册数据是否满足此协议

/// </summary>

/// <param name="pack"></param>

/// <param name="len"></param>

/// <returns></returns>

public static bool CheckProtocolByRegisterData(byte[] pack, int len)

{

bool flag = true;

try

{

int i;

for (i = 0; i < len; i++)

{

if (pack[i] == HEAD) //先找到头

{

break;

}

}

byte type = pack[i + 1];

if (type != DTU\_REGISTER\_DATA) //再判断时否是ZG\_DTU的注册类型

flag = false;

int high = pack[i + 2]; //高8位

int low = pack[i + 3]; //低8位

int length = high \* 256 + low; //从手机号码到最后的字节数

if (pack[i + length - 1] != END) //再找到尾

{

flag = false;

}

}

catch (Exception ex)

{

LogManager.AddError(ex);

flag = false;

}

return flag;

}

/// <summary>

/// 拆包

/// </summary>

/// <param name="socket"></param>

/// <param name="pack"></param>

/// <param name="packLen"></param>

/// <param name="content"></param>

/// <param name="conLen"></param>

/// <param name="type"></param>

/// <param name="tel"></param>

/// <returns></returns>

public static bool UnPack(byte[] pack, int packLen, ref byte[] content, ref int conLen, ref byte type, ref string tel)

{

try

{

int i;

for (i = 0; i < packLen; i++)

{

if (pack[i] == HD\_DTU.HEAD)

{

break;

}

}

type = pack[i + 1]; //类型

if (type == HD\_DTU.DTU\_REGISTER\_DATA) //注册数据拆包和

{

//解析接收到的数据

int high = pack[i + 2]; //高8位

int low = pack[i + 3]; //低8位

int length = high \* 256 + low; //从手机号码到最后的字节数

byte[] b\_tel = new byte[11];

Buffer.BlockCopy(pack, i + 4, b\_tel, 0, 11);

tel = Encoding.ASCII.GetString(b\_tel);

}

else if (type == HD\_DTU.DTU\_User\_Data) //用户数据拆包

{

//解析接收到的数据

int high = pack[i + 2]; //高8位

int low = pack[i + 3]; //低8位

int length = high \* 256 + low; //从手机号码到最后的字节数

byte[] b\_tel = new byte[11];

Buffer.BlockCopy(pack, i + 4, b\_tel, 0, 11);

tel = Encoding.ASCII.GetString(b\_tel);

conLen = length - 16;

Buffer.BlockCopy(pack, i + 4 + 11, content, 0, conLen);

}

else

{

//日志

string str = "";

for (int j = 0; j < packLen; j++)

{

str += pack[j] + " ";

}

LogManager.AddDebug("收到宏电DTU未知类型的数据:" + str);

}

}

catch (Exception ex)

{

LogManager.AddError(ex);

return false;

}

return true;

}

/// <summary>

/// 当收到注册信息无误后, 应调用此方法回应DTU

/// </summary>

public static bool ResponseRegister(Socket socket, string tel)

{

try

{

//应答客户端

byte[] sendBytes = new byte[16];

sendBytes[0] = HD\_DTU.HEAD;

sendBytes[1] = HD\_DTU.DSC\_REGISTER\_DATA;

sendBytes[2] = 0;

sendBytes[3] = 0x10;

byte[] b\_tel = Encoding.ASCII.GetBytes(tel);

Buffer.BlockCopy(b\_tel, 0, sendBytes, 4, 11);

sendBytes[4 + 11] = HD\_DTU.END;

int n = Send(socket, sendBytes, 16);

return n > 0;

}

catch (Exception ex)

{

LogManager.AddError(ex);

return false;

}

}

}

# SysConfig

/// <summary>

/// 读取与保存系统所有配置文件的类

/// </summary>

public class SysConfig

{

/// <summary>

/// 配置文件的路径

/// </summary>

private static readonly string SysConfigPath = AppDomain.CurrentDomain.BaseDirectory + "../../Config/SysConfig.xml";

private static readonly string ClientConfigPath = AppDomain.CurrentDomain.BaseDirectory + "../../Config/ClientConfig.xml";

public static UserProfile userProfile = new UserProfile(SysConfigPath);

public static OrgConfig orgConfig = new OrgConfig(SysConfigPath);

public static DTU\_StationConfig DTU\_StationConfig = new DTU\_StationConfig(SysConfigPath);

public static DBCJConfig DBCJconfig=new DBCJConfig(SysConfigPath);

public static ClientConfig1 clientConfig = new ClientConfig1(ClientConfigPath);

/// <summary>

/// 读取配置文件数据

/// </summary>

public static void ReadConfig()

{

userProfile.ReadConfig();

orgConfig.ReadConfig();

DTU\_StationConfig.ReadConfig();

DBCJconfig.ReadConfig();

clientConfig.ReadConfig();

}

/// <summary>

/// 保存配置到XML文件

/// </summary>

public static void SaveConfig()

{

userProfile.SaveConfig();

orgConfig.SaveConfig();

DTU\_StationConfig.SaveConfig();

DBCJconfig.SaveConfig();

clientConfig.SaveConfig();

}

}