添加头文件：

#include <WinSock2.h>

#include <windows.h>

# 通过注册表获取当前的串口名

//============自动寻找串口函数================================= //

//函数功能:通过扫描注册表来找出当前所有物理串口

//输入参数:无

//返回类型:无

//说 明:若搜索成功,则每搜到一个串口便发送消息通知主对话框,并将串口号以WPARAM传递

void FindComm()

{

//枚举当前系统中的串口

LONG result = 0;

HKEY key = NULL;

result = RegOpenKeyEx(HKEY\_LOCAL\_MACHINE, //需要打开的主键的名称

"HARDWARE\\DEVICEMAP\\SERIALCOMM",

//需要打开的子键的名称,设备串口

0, //保留，必须设置为0

KEY\_READ, //安全访问标记，也就是权限

&key); //得到的将要打开键的句柄，当不再需要句柄，

//必须调用 RegCloseKey 关闭它

if( result )

{

AfxMessageBox("无法获取串口，请确认是否安装并连接串口!");

return;

}

TCHAR portname[250]; //串口名

TCHAR data[250];

DWORD portnamelen = 0; //串口名长度

DWORD datalen = 0;

int index = 0;

while(1) //找完COM后跳出

{

portnamelen = 255;

datalen = 255;

result = RegEnumValue(key,

//Long，一个已打开项的句柄，或者指定一个标准项名

index++,

//Long，欲获取值的索引。注意第一个值的索引编号为零

portname,

//String，用于装载位于指定索引处值名的一个缓冲区

&portnamelen,

//Long，用于装载lpValueName缓冲区长度的一个变量。

//一旦返回，它会设为实际载入缓冲区的字符数量

NULL,

//Long，未用；设为零

NULL,

//Long，用于装载值的类型代码的变量

(LPBYTE)data, //Byte，用于装载值数据的一个缓冲区

&datalen); //Long，用于装载lpData缓冲区长度的一个变量。

//一旦返回，它会设为实际载入缓冲区的字符数量

if( result )

break;

//发送消息,WM\_USER+1为自定义消息,即找到串口的,并将串口号"COMx"通过WPARA M参数传送给主对话框窗口

//::AfxGetMainWnd()->m\_hWnd,获得主对话框句柄

//(WPARAM)(LPCTSTR)data,类型转换

::SendMessage(::AfxGetMainWnd()->m\_hWnd,WM\_FOUNDCOMM,(WPARAM)(LPCTSTR)data,0);

}

RegCloseKey(key); //调用 RegCloseKey 关闭打开键的句柄

}

# 打开串口

//==========串口打开函数===========================

//功 能:打开串口,将已打开的串口句柄赋值给hCom,给出串口打开状态ComIsOK,完成串口状态 设置

//输入参数:波特率,数据位,停止位,校验位

//返回类型:无

void OpenComm(int nBaud, int nData, int nStop, int nCal) {

hCom = CreateFile(strcomname, //串口号

GENERIC\_READ | GENERIC\_WRITE, //允许读或写

0, //独占方式

NULL,

OPEN\_EXISTING, //打开而不是创建

FILE\_ATTRIBUTE\_NORMAL | FILE\_FLAG\_OVERLAPPED,//重叠方式,用于异步通信

NULL );

if (hCom == INVALID\_HANDLE\_VALUE)

{

AfxMessageBox(\_T("打开COM失败，串口不存在或已被占用!"));

ComIsOK = false; return;

}

ComIsOK = true;

SetCommMask(hCom, EV\_TXEMPTY | EV\_RXCHAR); //设置事件掩码,暂时没用上

SetupComm(hCom,1024,1024); //设置输入缓冲区和输出缓冲区的大小都是1024

COMMTIMEOUTS TimeOuts; //设定读超时

TimeOuts.ReadIntervalTimeout = MAXDWORD;

TimeOuts.ReadTotalTimeoutConstant = 0;

TimeOuts.ReadTotalTimeoutMultiplier = 0; //设定写超时

TimeOuts.WriteTotalTimeoutConstant = 500;

TimeOuts.WriteTotalTimeoutMultiplier = 100;

if(SetCommTimeouts(hCom,&TimeOuts) == false)

{

CloseHandle(hCom);

ComIsOK = false; return;

} //串口属性配置

DCB dcb;

GetCommState(hCom,&dcb);

dcb.BaudRate=nBaud; //dcb.BaudRate=9600; //波特率为9600

dcb.ByteSize=nData; //dcb.ByteSize=8; //每个字节为8位

dcb.StopBits=nStop; //dcb.StopBits=ONESTOPBIT; //1位停止位

dcb.Parity=nCal; //dcb.Parity=NOPARITY; //无奇偶检验位

SetCommState(hCom, &dcb);

PurgeComm(hCom, PURGE\_TXCLEAR | PURGE\_RXCLEAR);

if (SetCommState(hCom, &dcb) == false)

{

CloseHandle(hCom);

ComIsOK = false;

return;

}

return;

}

# 关闭串口

1. *//==========串口关闭控制函数=====================*
2. void CloseComm()
3. {
4. CloseHandle(hCom);
5. hCom = NULL;
6. ComIsOK = false;
7. }

# 4、函数原型

1、打开串口返回操作句柄

HANDLE CreateFileA(portname, //串口名

GENERIC\_READ | GENERIC\_WRITE, //支持读写

0, //独占方式，串口不支持共享

NULL,//安全属性指针，默认值为NULL

OPEN\_EXISTING, //打开现有的串口文件

0, //0：同步方式，FILE\_FLAG\_OVERLAPPED：异步方式

NULL);//用于复制文件句柄，默认值为NULL，对串口而言该参数必须置为NULL

返回 （HANDLE）-1 则打开失败

2、

# 4、生肉代消化

#include "readport.h"

#include <stdio.h>

#include <string.h>

#include <WinSock2.h>

#include <windows.h>

WZSerialPort::WZSerialPort()

{

}

WZSerialPort::~WZSerialPort()

{

}

bool WZSerialPort::open(const char\* portname,

int baudrate,

char parity,

char databit,

char stopbit,

char synchronizeflag)

{

this->synchronizeflag = synchronizeflag;

HANDLE hCom = NULL;

if (this->synchronizeflag)

{

//同步方式

hCom = CreateFileA(portname, //串口名

GENERIC\_READ | GENERIC\_WRITE, //支持读写

0, //独占方式，串口不支持共享

NULL,//安全属性指针，默认值为NULL

OPEN\_EXISTING, //打开现有的串口文件

0, //0：同步方式，FILE\_FLAG\_OVERLAPPED：异步方式

NULL);//用于复制文件句柄，默认值为NULL，对串口而言该参数必须置为NULL

}

else

{

//异步方式

hCom = CreateFileA(portname, //串口名

GENERIC\_READ | GENERIC\_WRITE, //支持读写

0, //独占方式，串口不支持共享

NULL,//安全属性指针，默认值为NULL

OPEN\_EXISTING, //打开现有的串口文件

FILE\_FLAG\_OVERLAPPED, //0：同步方式，FILE\_FLAG\_OVERLAPPED：异步方式

NULL);//用于复制文件句柄，默认值为NULL，对串口而言该参数必须置为NULL

}

if (hCom == (HANDLE)-1)

{

return false;

}

//配置缓冲区大小

if (!SetupComm(hCom, 1024, 1024))

{

return false;

}

// 配置参数

DCB p;

memset(&p, 0, sizeof(p));

p.DCBlength = sizeof(p);

p.BaudRate = baudrate; // 波特率

p.ByteSize = databit; // 数据位

switch (parity) //校验位

{

case 0:

p.Parity = NOPARITY; //无校验

break;

case 1:

p.Parity = ODDPARITY; //奇校验

break;

case 2:

p.Parity = EVENPARITY; //偶校验

break;

case 3:

p.Parity = MARKPARITY; //标记校验

break;

}

switch (stopbit) //停止位

{

case 1:

p.StopBits = ONESTOPBIT; //1位停止位

break;

case 2:

p.StopBits = TWOSTOPBITS; //2位停止位

break;

case 3:

p.StopBits = ONE5STOPBITS; //1.5位停止位

break;

}

if (!SetCommState(hCom, &p))

{

// 设置参数失败

return false;

}

//超时处理,单位：毫秒

//总超时＝时间系数×读或写的字符数＋时间常量

COMMTIMEOUTS TimeOuts;

TimeOuts.ReadIntervalTimeout = 1000; //读间隔超时

TimeOuts.ReadTotalTimeoutMultiplier = 500; //读时间系数

TimeOuts.ReadTotalTimeoutConstant = 5000; //读时间常量

TimeOuts.WriteTotalTimeoutMultiplier = 500; // 写时间系数

TimeOuts.WriteTotalTimeoutConstant = 2000; //写时间常量

SetCommTimeouts(hCom, &TimeOuts);

PurgeComm(hCom, PURGE\_TXCLEAR | PURGE\_RXCLEAR);//清空串口缓冲区

memcpy(pHandle, &hCom, sizeof(hCom));// 保存句柄

return true;

}

void WZSerialPort::close()

{

HANDLE hCom = \*(HANDLE\*)pHandle;

CloseHandle(hCom);

}

int WZSerialPort::send(string dat)

{

HANDLE hCom = \*(HANDLE\*)pHandle;

if (this->synchronizeflag)

{

// 同步方式

DWORD dwBytesWrite = dat.length(); //成功写入的数据字节数

BOOL bWriteStat = WriteFile(hCom, //串口句柄

(char\*)dat.c\_str(), //数据首地址

dwBytesWrite, //要发送的数据字节数

&dwBytesWrite, //DWORD\*，用来接收返回成功发送的数据字节数

NULL); //NULL为同步发送，OVERLAPPED\*为异步发送

if (!bWriteStat)

{

return 0;

}

return dwBytesWrite;

}

else

{

//异步方式

DWORD dwBytesWrite = dat.length(); //成功写入的数据字节数

DWORD dwErrorFlags; //错误标志

COMSTAT comStat; //通讯状态

OVERLAPPED m\_osWrite; //异步输入输出结构体

//创建一个用于OVERLAPPED的事件处理，不会真正用到，但系统要求这么做

memset(&m\_osWrite, 0, sizeof(m\_osWrite));

m\_osWrite.hEvent = CreateEvent(NULL, TRUE, FALSE, L"WriteEvent");

ClearCommError(hCom, &dwErrorFlags, &comStat); //清除通讯错误，获得设备当前状态

BOOL bWriteStat = WriteFile(hCom, //串口句柄

(char\*)dat.c\_str(), //数据首地址

dwBytesWrite, //要发送的数据字节数

&dwBytesWrite, //DWORD\*，用来接收返回成功发送的数据字节数

&m\_osWrite); //NULL为同步发送，OVERLAPPED\*为异步发送

if (!bWriteStat)

{

if (GetLastError() == ERROR\_IO\_PENDING) //如果串口正在写入

{

WaitForSingleObject(m\_osWrite.hEvent, 1000); //等待写入事件1秒钟

}

else

{

ClearCommError(hCom, &dwErrorFlags, &comStat); //清除通讯错误

CloseHandle(m\_osWrite.hEvent); //关闭并释放hEvent内存

return 0;

}

}

return dwBytesWrite;

}

}

string WZSerialPort::receive()

{

HANDLE hCom = \*(HANDLE\*)pHandle;

string rec\_str = "";

char buf[1024];

if (this->synchronizeflag)

{

//同步方式

DWORD wCount=1024; //成功读取的数据字节数

BOOL bReadStat = ReadFile(hCom, //串口句柄

buf, //数据首地址

wCount, //要读取的数据最大字节数

&wCount, //DWORD\*,用来接收返回成功读取的数据字节数

NULL); //NULL为同步发送，OVERLAPPED\*为异步发送

for (int i = 0; i < strlen(buf); i++)

{

if (buf[i] != -52)

{

//cout << buf[i];

rec\_str += buf[i];

//revcmsg.push\_back(buf[i]);

}

else

{

break;

}

}

return rec\_str;

}

else

{

//异步方式

DWORD wCount = 1024; //成功读取的数据字节数

DWORD dwErrorFlags; //错误标志

COMSTAT comStat; //通讯状态

OVERLAPPED m\_osRead; //异步输入输出结构体

//创建一个用于OVERLAPPED的事件处理，不会真正用到，但系统要求这么做

memset(&m\_osRead, 0, sizeof(m\_osRead));

m\_osRead.hEvent = CreateEvent(NULL, TRUE, FALSE, L"ReadEvent");

ClearCommError(hCom, &dwErrorFlags, &comStat); //清除通讯错误，获得设备当前状态

if (!comStat.cbInQue)

return ""; //如果输入缓冲区字节数为0，则返回false

//std::cout << comStat.cbInQue << std::endl;

BOOL bReadStat = ReadFile(hCom, //串口句柄

buf, //数据首地址

wCount, //要读取的数据最大字节数

&wCount, //DWORD\*,用来接收返回成功读取的数据字节数

&m\_osRead); //NULL为同步发送，OVERLAPPED\*为异步发送

if (!bReadStat)

{

if (GetLastError() == ERROR\_IO\_PENDING) //如果串口正在读取中

{

//GetOverlappedResult函数的最后一个参数设为TRUE

//函数会一直等待，直到读操作完成或由于错误而返回

GetOverlappedResult(hCom, &m\_osRead, &wCount, TRUE);

}

else

{

ClearCommError(hCom, &dwErrorFlags, &comStat); //清除通讯错误

CloseHandle(m\_osRead.hEvent); //关闭并释放hEvent的内存

return "";

}

}

for (int i = 0; i < strlen(buf); i++)

{

if (buf[i] != -52)

{

rec\_str += buf[i];

//revcmsg.push\_back(buf[i]);

}

else

{

break;

}

}

return rec\_str;

}

}