# VC

## 数据类型

### Windows数据类型

BYTE是1个字节（char），8位

（0[0x00]到255[0xff],能指向256个bit位，也就是256Bit内存大小）

WORD是2个字节（Unsigned short），16位（0到65535）

DWORD是4个字节（Unsigned long）,32位（0到232-1)【可以指向4GB内存】

HANDLE唯一标识一个对象（void \*）

### Unicode和多字节

#### 多字节

char c=’A’;

char \*p=”hello”;

char a[10];

char a[]=”hello”;

字符串函数：

strlen

#### Unicode

typedef unsigned short wchar\_t;

wchar\_t c=’A’;

wchar\_t \*p=L”hello”;

wchar\_t a[]=L”hello”;

字符串函数:

wcslen

#### 综合方案

#ifndef UNICODE

typedef WCHAR TCHAR, \*PTCHAR;

typedef LPWSTR LPTCH,PTSTR,LPTSTR;

typedef LPCWSTR LPCTSTR;

#define TEXT(quote) L##quote //或者是\_T

#else

typedef char TCHAR, \*PTCHAR;

typedef LPSTR LPTCH,PTSTR,LPTSTR;

typedef LPCSTR LPCTSTR;

#define TEXT(quote) quote

#elseif

因为有了这样的定义，所以在使用时，可以直接用TCHAR作为基本的字符数据类型，在写字符串时可以写\_T(“hello”)。

#ifdef UNICODE

#define MessageBox MessageBoxW

#else

#define MessageBox MessageBoxA

#endif

所有和文字处理相关的函数都有W和A版本，可以显示调用，也可以直接调用函数原型，通过以上的定义自动转换为需要的类型。

Windows中定义的通用版基本字符串处理函数：

lstrlen, lstrcpy, lstrcpyn, lstrcat, lstrcmp, lstrcmpi

\_stprintf

### LPARAM

DWORD x=100;

DWORD y=200;

LPARAM lParam=MAKELPARAM(x,y);

## WindowsThread

### 线程

DWORD WINAPI fun(LPVOID param)

{

int x = \*(int\*)param;

cout << x << endl;

return 0;

}

//创建并运行线程

int param = 10;

HANDLE handle = CreateThread(NULL, 0, fun, &param, 0, NULL);

//创建使不运行，程序挂起

HANDLE handle = CreateThread(NULL, 0, fun, &param, CREATE\_SUSPENDED, NULL);

ResumeThread(handle);//恢复运行

//等待线程运行结束

WaitForSingleObject(handle, INFINITE);

//关闭线程

TerminateThread(handle, 0);

CloseHandle(handle);//句柄资源需要释放

//挂起、恢复线程

SuspendThread(handle);

ResumeThread(handle);

对于一个线程，挂起操作几次就要恢复操作几次

SuspendThread，ResumeThread返回值是该线程剩余挂起次数，-1代表线程为NULL

//确保线程重启

while (true) if (ResumeThread(thr) <= 0) break;

#include <process.h>

UINT threadId;

(HANDLE)\_beginthreadex(NULL, 0, ListenThread, NULL, 0, &threadId);

UINT WINAPI ListenThread(void\* pParam)

### 线程同步

当对象被占用时WAIT\_TIMEOUT == WaitForSingleObject(h,0)

WaitForSingleObject(h, INFINITE)无限制等待，执行成功后占用对象

#### 互斥对象

互斥对象上几次锁就要调用几次ReleaseMutex。上锁行为包括创建锁时的占用，以及执行WaitForSingleObject成功。

HANDLE hMutex; //要被多个线程调用

hMutex=CreateMutex(NULL,FALSE,NULL);//初始化，最后一个参数代表是无名的互斥对象，FALSE代表当前线程创建锁时互斥对象没被占用（如果为TRUE，表示创建的同时占用互斥对象），也就是其它线程可以使用。

::WaitForSingleObject(hMutex,INFINITE);//运行线程时，先运行这一句，每个线程前面都有这一句，就可以保证其中一个线程没结束，或者没有调用ReleaseMutex函数，其它线程代码不能运行，INFINITE代表超时值为一直等待

ReleaseMutex(hMutex); //当线程运行时间久，不同线程交替运行，但只是保护循环中某处的代码时可以用这个，而不用等待线程结束自动释放互斥对象

可以使用有名互斥对象判断进程是否已经执行

if(CreateMutex(NULL,TRUE,"xx"))

if(ERROR\_ALREADY\_EXISTS==GetLastError()){已经执行}

#### 关键代码段

类似互斥对象和事件对象，这两个对象是内核对象，速度较慢。关键代码段工作在用户方式下，速度快，但无法设置超时值，容易造成死锁。

CRITICAL\_SECTION cs;//要被多个线程调用，所以一般为全局变量

InitializeCriticalSection(&cs);//创建

DeleteCriticalSection(&cs);//释放

线程使用时:

EnterCriticalSection(&cs);

。。。。。。。要保护的代码内容。。。。。

LeaveCriticalSection(&cs);

//线程结束不会自动释放，必须要用代码

#### hEvent

HANDLE hEvent; hEvent=CreateEvent(NULL,TRUE,FALSE,NULL);//TRUE代表创建的是手动事件，FALSE代表创建时未占用（ResetEvent状态,内核占用该对象）

SetEvent将对象从内核拿出并占用

ResetEvent将对象还给内核

多次调用SetEvent(hEvent)不会有影响

##### 事件控制

DWORD WINAPI fun(LPVOID p)

{

while (true)

{

WaitForSingleObject(hEvent, INFINITE);

//要运行的代码

ResetEvent(hEvent);

}

return 0;

}

判断处于ResetEvent状态（说明已经执行到线程末尾的ResetEvent处，线程处于空闲状态），然后再控制线程开始运行：

if (WAIT\_TIMEOUT == WaitForSingleObject(hEvent, 0))

{ SetEvent(hEvent); }

WAIT\_OBJECT\_0 ：核心对象已被激活

WAIT\_TIMEOUT ：等待超时

WAIT\_FAILED ：出现错误

##### 信号控制

可以被多个线程调用

相当于bool值,通过HasSignal()判断有无信号

struct Signal

{

Signal()

{

m\_hEvent = CreateEvent(NULL, TRUE, FALSE, NULL);

}

bool HasSignal()

{

return WAIT\_TIMEOUT != WaitForSingleObject(m\_hEvent,0);

}

void SetSignal()

{

SetEvent(m\_hEvent);

}

void DelSignal()

{

ResetEvent(m\_hEvent);

}

HANDLE m\_hEvent;

};

## WindowsNetWork

### Socket

Send等函数返回值为==SOCKET\_ERROR，代表发送失败。

Socket是单向的，服务器和客户端各自维护一个文件。

该文件可以由本端写入，另一端读取。

本端是在内存区域不断放入，另一端是不断从内存区域拿走。

#### 初始化

#include <windows.h>//#include <Winsock2.h>

#pragma comment(lib,"ws2\_32.lib")

WSADATA wsaData;

WORD wVersionRequested = MAKEWORD( 2, 2 );

if (0 != WSAStartup( wVersionRequested, &wsaData ))

{

return;

}

if ( LOBYTE( wsaData.wVersion ) != 2 ||HIBYTE( wsaData.wVersion ) != 2 )

{

WSACleanup( );//清理数据

return;

}

#### TCP

##### Server

//创建socket

SOCKET sockSrv=socket(AF\_INET,SOCK\_STREAM,0);

SOCKADDR\_IN addrSrv;

addrSrv.sin\_addr.S\_un.S\_addr=htonl(INADDR\_ANY);

addrSrv.sin\_family=AF\_INET;

addrSrv.sin\_port=htons(5555); //端口号

bind(sockSrv,(SOCKADDR\*)&addrSrv,sizeof(SOCKADDR));

listen(sockSrv,5); //最多能接受5个客户端链接，如果是SOMAXCONN，则自动设置为合理的最大值

while(1)

{

SOCKADDR\_IN addrClient;

int len=sizeof(SOCKADDR);

//接受链接

SOCKET sockConn=accept(sockSrv,(SOCKADDR\*)&addrClient,&len);

//发送

char sendBuf[100]={0};

sprintf(sendBuf,"%s",inet\_ntoa(addrClient.sin\_addr));

send(sockConn,sendBuf,strlen(sendBuf)+1,0);

//接收

const int bufSize = 100;

char recvBuf[bufSize ] = {0};

recv(sockConn,recvBuf,bufSize ,0);//返回值为接收到的实际大小,网络错误时，返回SOCKET\_ERROR

printf("%s\n",recvBuf);

//关闭

closesocket(sockConn);

}

##### Client

客户端在第二次连接时，需要重新定义SOCKET sockClient

SOCKET sockClient=socket(AF\_INET,SOCK\_STREAM,0);

SOCKADDR\_IN addrSrv;

addrSrv.sin\_addr.S\_un.S\_addr=inet\_addr("127.0.0.1");

addrSrv.sin\_family=AF\_INET;

addrSrv.sin\_port=htons(5555);

if(SOCKET\_ERROR==connect(sockClient,(SOCKADDR\*)&addrSrv,sizeof(SOCKADDR)))

{return ;}

//接收

const int bufSize = 100;

char recvBuf[bufSize ]={0};

recv(sockClient,recvBuf,bufSize ,0);

printf("%s\n",recvBuf);

//发送

send(sockClient,"clientSend",strlen("client")+1,0);

//关闭

closesocket(sockClient);

//WSACleanup();

#### UDP

##### Server

SOCKET sockSrv=socket(AF\_INET,SOCK\_DGRAM,0);

SOCKADDR\_IN addrSrv;

addrSrv.sin\_addr.S\_un.S\_addr=htonl(INADDR\_ANY);

addrSrv.sin\_family=AF\_INET;

addrSrv.sin\_port=htons(5555);

bind(sockSrv,(SOCKADDR\*)&addrSrv,sizeof(SOCKADDR));

//不调用listen函数

SOCKADDR\_IN addrClient;

int len=sizeof(SOCKADDR);

char recvBuf[100];

//接收

recvfrom(sockSrv,recvBuf,100,0,(SOCKADDR\*)&addrClient,&len);

printf("%s\n",recvBuf);

closesocket(sockSrv);

WSACleanup();

##### Client

SOCKET sockClient=socket(AF\_INET,SOCK\_DGRAM,0);

SOCKADDR\_IN addrSrv;

addrSrv.sin\_addr.S\_un.S\_addr=inet\_addr("127.0.0.1");

addrSrv.sin\_family=AF\_INET;

addrSrv.sin\_port=htons(5555);

//不调用connect函数

sendto(sockClient,"client",strlen("clinet")+1,0,(SOCKADDR\*)&addrSrv,sizeof(SOCKADDR));

closesocket(sockClient);

WSACleanup();

#### 文件传输

##### 客户端（发送）

string fileName = "1.jpg";

string filePath = "D:\\";

ifstream ifile(filePath + fileName,ifstream::binary);

ifile.seekg(0, ios::end);

std::streamoff fileSize = ifile.tellg();

ifile.seekg(0,ios::beg);

if (!ifile) return;

stringstream ss;

ss << fileName << " " << fileSize;

string fileInfo = ss.str();

send(sockClient, fileInfo.c\_str(), fileInfo.size() + 1, 0);//发送文件名及文件大小

const int BufferSize = 1024;

int readSize = 0;

char SendBuf[BufferSize] = {0};

while (true)

{

ifile.read(SendBuf, BufferSize);

readSize = (int)ifile.gcount();

if (readSize == 0) break;

send(sockClient, SendBuf, readSize, 0);

}

ifile.close();

//等待服务器接收成功

char EndMessage[1024];

recv(sockClient, EndMessage, 1024, 0);

cout << EndMessage << endl;

##### 服务器（接收）

int recSize = 0;

//获取文件名和大小

char FileInfo[1024] = { 0 };

recSize = recv(sockConn, FileInfo, 1024, 0);

if (recSize == 0) return;

stringstream ss(FileInfo);

string FileName;

std::streamoff fileSize = 0;

ss >> FileName >> fileSize;

//创建文件

string filePath = "E:\\";

filePath += FileName;

ofstream ofile(filePath,ofstream::binary);

if (!ofile) return;

//获取文件内容

std::streamoff RecvfileSize = 0;

const int BufferSize = 1024;

char RecvBuff[BufferSize] = {0};

while (true)

{

recSize = recv(sockConn, RecvBuff, BufferSize, 0);

if (recSize == SOCKET\_ERROR) { cout << "erro " << endl; return; }

ofile.write(RecvBuff, recSize);

RecvfileSize += recSize;

if (RecvfileSize == fileSize) break;//接收完文件

}

send(sockConn, "suc", 4, 0);

ofile.close();

#### 检测socket是否可写

timeval tm = {0,100};

fd\_set fd\_write;

FD\_ZERO(&fd\_write);

FD\_SET(\*pSocket,&fd\_write);

int iSelEro = select(\*pSocket + 1 ,NULL,&fd\_write,NULL,&tm);

### 网络基本应用

#### 获取主机名/IP地址

需要初始化socket

char\* name=new char[50];

gethostname(name,50);

//通过主机名称获取ip地址

char\* ip=new char[50];

struct hostent \*pHostent=gethostbyname(name);

if(pHostent!=NULL)

{

char\* lpAddr=pHostent->h\_addr\_list[0];

if(lpAddr!=NULL)

{

struct in\_addr inAddr;

memmove(&inAddr,lpAddr,4);

ip=inet\_ntoa(inAddr);//转换成标准的ip地址形式

}

}

#### 获取MAC

#include <IPHlpApi.h>

#pragma comment(lib,"Iphlpapi.lib")

DWORD AdapterInfoSize=0;

DWORD Err=GetAdaptersInfo(NULL,&AdapterInfoSize);

if(Err!=0&&Err!=ERROR\_BUFFER\_OVERFLOW)

{

TRACE("获取网卡信息失败");

return ;

}

//分配网卡信息内存

PIP\_ADAPTER\_INFO pAdapterInfo=(PIP\_ADAPTER\_INFO)GlobalAlloc(GPTR,AdapterInfoSize);

if(pAdapterInfo==NULL)

{

TRACE("分配网卡信息内存失败");

return ;

}

if(GetAdaptersInfo(pAdapterInfo,&AdapterInfoSize)!=0)

{

TRACE("获取网卡信息失败");

GlobalFree(pAdapterInfo);

return ;

}

CString mac;

mac.Format("%02X-%02X-%02X-%02X-%02X-%02X",pAdapterInfo->Address[0],pAdapterInfo->Address[1],pAdapterInfo->Address[2],pAdapterInfo->Address[3],pAdapterInfo->Address[4],pAdapterInfo->Address[5]);

#### 获取远程主机MAC

#include <IPHlpApi.h>

#pragma comment(lib,"Iphlpapi.lib")

char \*addr ="192.168.10.24";

u\_char mac[6]={0xff,0xff,0xff,0xff,0xff,0xff};

ULONG ulen=6;

if(::SendARP(::inet\_addr(addr),0,(ULONG\*)mac,&ulen)==NO\_ERROR)

{

CString str;

str.Format("%02X-%02X-%02X-%02X-%02X-%02X",mac[0],mac[1],mac[2],mac[3],mac[4],mac[5]);

}

## 窗口

### 程序结构

结构：

Winmain（…..）

{

…………….

注册窗口类

CreateWindowEx(……窗口类….窗口过程函数……..)

While(消息循环){……}

}

### 对话框

#### 程序结构

INT\_PTR CALLBACK DlgProc(HWND hDlg, UINT msg, WPARAM wParam, LPARAM lParam);

HINSTANCE hgInst;

int WINAPI WinMain(HINSTANCE hThisApp, HINSTANCE hPrevApp, LPSTR lpCmd, int nShow)

{

hgInst = hThisApp;

HWND hwnd = CreateDialog(hThisApp, MAKEINTRESOURCE(IDD\_DLG), GetDesktopWindow(), (DLGPROC)DlgProc);

if (!hwnd) return 0;

ShowWindow(hwnd, nShow);

MSG msg;

while (GetMessage(&msg, NULL, 0, 0))

{

TranslateMessage(&msg);

DispatchMessage(&msg);

}

return 0;

}

INT\_PTR CALLBACK DlgProc(HWND hwnd, UINT msg, WPARAM wParam, LPARAM lParam)

{

switch (msg)

{

case WM\_INITDIALOG:

SendMessage(hwnd, WM\_SETICON, ICON\_SMALL, (LPARAM)LoadIcon(hgInst, MAKEINTRESOURCE(IDI\_ICON)));

break;

case WM\_DESTROY:

PostQuitMessage(0);

break;

default:

return DefWindowProc(hwnd, msg, wParam, lParam);

}

return 0;

}

#### 获取控件句柄

HWND wnd1 = GetDlgItem(hwnd, IDC\_View0);

### 资源加载

#### 对话框窗口

DialogBox(hInstance, MAKEINTRESOURCE(IDD\_DLG), hWnd, msgProcFun);

#### 字符串资源

LoadStringW(hInstance, IDS\_APP\_TITLE, szTitle, 100);

#### 菜单

MAKEINTRESOURCEW(IDC\_MENU);

对应：WM\_COMMAND

int cmdId = LOWORD(wParam);

#### 快捷快捷键

HACCEL hAccelTable = LoadAccelerators(hInstance, MAKEINTRESOURCE(IDC\_ACC));

对应TranslateAccelerator函数

### 外观

#### 去除边框

HWND hWnd = CreateWindowW(szWindowClass, szTitle, WS\_VISIBLE,

CW\_USEDEFAULT, 0, CW\_USEDEFAULT, 0, nullptr, nullptr, hInstance, nullptr);

SetWindowLong(hWnd, GWL\_STYLE, GetWindowLong(hWnd, GWL\_STYLE) & ~WS\_CAPTION);

#### 全屏

方法一：

ShowWindow(hWnd, SW\_MAXIMIZE);

方法二：

int cx = GetSystemMetrics(SM\_CXSCREEN);

int cy = GetSystemMetrics(SM\_CYSCREEN);

MoveWindow(hWnd, 0, 0, cx, cy, TRUE);

或:

HWND hWnd = CreateWindowW(szWindowClass, szTitle, WS\_VISIBLE,

CW\_USEDEFAULT, 0, cx, cy,

nullptr, nullptr, hInstance, nullptr);

### 背景色

wcex.hbrBackground = CreateSolidBrush(RGB(0, 0, 0));

### 双击功能

注册窗口类时加入属性CS\_DBLCLKS

### 控件

#### 滚动条

HWND hWnd = CreateWindowW(szWindowClass, szTitle, WS\_OVERLAPPEDWINDOW|WS\_VSCROLL,

CW\_USEDEFAULT, 0, CW\_USEDEFAULT, 0, nullptr, nullptr, hInstance, nullptr);

#### 按钮

HWND BtnWnd = 0;

BtnWnd = CreateWindow(TEXT("BUTTON"), TEXT(""), WS\_VISIBLE | WS\_CHILD | BS\_DEFPUSHBUTTON,

10, 10, 100, 30,

hWnd, NULL, (HINSTANCE)GetWindowLong(hWnd, GWL\_HINSTANCE), NULL);

WM\_COMMAND

if (lParam == (UINT)BtnWnd)

{

MessageBox(hWnd, TEXT("Test"), TEXT("Test"), MB\_OK);

}

#### 静态框

CreateWindow(TEXT("STATIC"), TEXT(""), WS\_VISIBLE | WS\_CHILD | SS\_BLACKFRAME,

10, 10, 100, 30,

hWnd, NULL, (HINSTANCE)GetWindowLong(hWnd, GWL\_HINSTANCE), NULL);

### 消息

#### WM\_SIZE

int cx = LOWORD(lParam);

int cy = HIWORD(lParam);

#### WM\_MOUSEWHEEL

int zDelta = GET\_WHEEL\_DELTA\_WPARAM(wParam);

#### WM\_VSCROLL

UINT nPos = HIWORD(wParam);

UINT nSBCode = LOWORD(wParam);

## 绘图

### 画刷

HBRUSH hbr = CreateSolidBrush(RGB(0, 0, 0));

HBRUSH hbr = (HBRUSH)GetStockObject(NULL\_BRUSH);//空画刷

### WM\_PAINT

PAINTSTRUCT ps={0};

HDC hdc=BeginPaint(hWnd,&ps); //该函数让无效区域背景被擦除

TextOut(hdc,x,y,"paint",5);

EndPaint(hWnd,&ps);

//BeginPaint和EndPaint函数成对结束后，窗口区域才变得有效

### WM\_PAINT外

HDC hdc=GetDC(hwnd);//如果hwnd为NULL，则代表桌面

TextOut(hdc,x,y,"paint",5);

ReleaseDC(hwnd,hdc);

//这对函数属于在某个指定区域绘图，不会让无效区域变得有效，也不一定是在无效区域绘图，不像WM\_PAINT时需要收集足够多的绘图资讯。要显示地使无效区域变得有效。

### 获取字体大小

TEXTMETRIC tm;

HDC hdc=::GetDC(hWnd);

GetTextMetrics(hdc,&tm);

ReleaseDC(hWnd,hdc);

可以从tm结构体中获取系统字体信息

## GDI+

### 初始化和释放

#include <comdef.h>

#include <GdiPlus.h>

#pragma comment(lib,"gdiplus.lib")

using namespace Gdiplus;

=======旧版本

#ifndef ULONG\_PTR

#define ULONG\_PTR unsigned long

#endif

初始化：

GdiplusStartupInput gdiplusStartupInput;

ULONG\_PTR pGdiToken;

GdiplusStartup(&pGdiToken,&gdiplusStartupInput,NULL);

释放：

if (pGdiToken>0) GdiplusShutdown(pGdiToken);

### Image

#### GetEncoderClsid

bool GetEncoderClsid(const WCHAR\* format, CLSID\* pClsid)

{

UINT num = 0;

UINT size = 0;

ImageCodecInfo\* pImageCodecInfo = NULL;

GetImageEncodersSize(&num, &size);

if(size == 0) return false;

pImageCodecInfo = (ImageCodecInfo\*)(malloc(size));

if(pImageCodecInfo == NULL) return false;

GetImageEncoders(num, size, pImageCodecInfo);

for(UINT index = 0; index < num; ++index)

{

if( wcscmp(pImageCodecInfo[index].MimeType, format) == 0 )

{

\*pClsid = pImageCodecInfo[index].Clsid;

free(pImageCodecInfo);

return true;

}

}

free(pImageCodecInfo);

return false;

}

用法：

CLSID encoderClsid

GetEncoderClsid(L"image/jpeg", &encoderClsid);

GetEncoderClsid(L"image/png", &encoderClsid);

GetEncoderClsid(L"image/bmp", &encoderClsid);

GetEncoderClsid(L"image/gif", &encoderClsid);

GetEncoderClsid(L"image/tiff", &encoderClsid);

#### 绘制图像

Image image(L"d:/1.jpg");

Graphics graphics(hdc);

graphics.DrawImage(&image,0,0,100,100);

#### 获取尺寸

int width=image.GetWidth();

int height=image.GetHeight();

#### 格式转换

Image img(L"D:\\b.bmp");

CLSID jpgClsid;

GetEncoderClsid(L"image/jpeg", &jpgClsid);

if( Ok != img.Save(L"D:\\xxx.jpg",&jpgClsid)){..}

#### 获取缩略图

Image \*pImg = img.GetThumbnailImage(50,50);

delete pImg;

### 颜色

第一个参数数透明度，0代表全透明，255代表不透明

Color drawClr(AlphaVal,GetRValue(clr),GetGValue(clr),GetBValue(clr));

### 画笔

Pen( const Color& color, REAL width);

Pen( const Brush\* brush, REAL width);

pen.SetDashStyle(DashStyleDash);//虚线

### 画刷

//实心画刷

SolidBrush brush(drawClr);

//阴影画刷 bkColor为透明时可以看到效果

HatchBrush brush(HatchStyleHorizontal, forColor, bkColor);

//渐变画刷

LinearGradientBrush brush(pointA, pointB, Color(AlphaVal,255,0,0),Color(AlphaVal,255,255,0));

//图像画刷

Image image(L"img.jpg");

TextureBrush tBrush1(&image);

### 绘制直线

#define WIDTH 2

Graphics graphics(hDC);

Pen newPen(drawClr,WIDTH);

graphics.DrawLine(&newPen,10,30,100,100);

### 绘制矩形

SolidBrush brush(Color::BurlyWood);

GetClientRect(rect);

graphics.FillRectangle(&brush,rect.left,rect.top,rect.right,rect.bottom);

//FillRectangle绘制实心矩形 DrawRectangle绘制矩形边框

### 绘制文字

SolidBrush brush(drawClr);

PointF pf(pt.x,pt.y);

Gdiplus::Font font(\_T("宋?体?"), 10);

Graphics graphics(MemDC.m\_hDC);

graphics.DrawString(name.GetBuffer(),name.GetLength(), &font,pf,&brush);

### bmp转成jpg流

CLSID encoderClsid;

GetEncoderClsid(L"image/jpeg", &encoderClsid);

Bitmap Bmp(hDib, NULL);//dib绑定到bmp后需要调用DeleteObject(hDib);

IStream \*pStream = NULL;

CreateStreamOnHGlobal(NULL, TRUE, &pStream);//创建一个IStream

Bmp.Save(pStream, &encoderClsid);//将bmp写入IStream

HGLOBAL hGlobal = NULL;

GetHGlobalFromStream(pStream, &hGlobal);//获取IStream的内存句柄

LPBYTE pBits = (LPBYTE)GlobalLock(hGlobal);//pBits就是存储jpg内容的内存指针

size\_t size = GlobalSize(pBits);

/////////////

DeleteObject(hDib);

pStream->Release();

//GlobalFree(hGlobal);//与pStream指向同一个内存，同时释放会访问冲突

=======================对jpg流进行使用

//显示

IStream \*pImgStream = NULL;

CreateStreamOnHGlobal(NULL, TRUE, &pImgStream);

ULONG Written = 0;

pImgStream->Write(pBits, size, &Written);

Image img(pImgStream);

CClientDC dc(this);

Graphics graphics(dc.m\_hDC);

graphics.DrawImage(&img, 0, 0, 100, 100);

pImgStream->Release();

======================= hDib

char chBmpBuf[2048] = { 0 };

BITMAPINFO \*pBmpInfo = (BITMAPINFO \*)chBmpBuf;

pBmpInfo->bmiHeader.biSize = sizeof(BITMAPINFOHEADER);

pBmpInfo->bmiHeader.biWidth = Width;//位图的宽度像素为单位

pBmpInfo->bmiHeader.biHeight = Height;//位图的高度

pBmpInfo->bmiHeader.biPlanes = 1;

pBmpInfo->bmiHeader.biBitCount = biBitCount;//每个像素的位数(必须是8的倍数)

pBmpInfo->bmiHeader.biCompression = BI\_RGB;//压缩方式，一般为0或者BI\_RGB（未压缩）

for (int i = 0;i<256;i++)

{

pBmpInfo->bmiColors[i].rgbBlue = i;

pBmpInfo->bmiColors[i].rgbGreen = i;

pBmpInfo->bmiColors[i].rgbRed = i;

pBmpInfo->bmiColors[i].rgbReserved = i;

}

BYTE\* PDIBBuffer = NULL;

HBITMAP hDib = CreateDIBSection(NULL, pBmpInfo, DIB\_RGB\_COLORS, (void\*\*)&PDIBBuffer, NULL, 0);//HDC仅在 usage参数为DIB\_PAL\_COLORS才使用

memcpy\_s(PDIBBuffer, bufSize, SrcBuffer, bufSize);

## Dll

### 动态库信息查看

Vs命令提示符查看dll依赖项

dumpbin -dependents d:\\*\*.dll

-headers 查看dll信息

-exports 查看导出函数

### 导出声明

#### 导出函数

extern "C" void \_declspec(dllexport) fun(){}

#### 导出类

class \_declspec(dllexport) CtestBase{};

#### stdcall

void \_\_stdcall Run()

def文件：

EXPORTS

Run

工程属性:

连接器->输入->模块定义文件-> xxx.def

使用：

int typedef (\_\_stdcall \*Fun)();

### 共享数据段

不同进程加载同一dll时候，可以共享相同的数据。

#pragma data\_seg ("shareData")//shareData为自定义的段名

HWND g\_hWnd=NULL;//共享数据必须初始化，否则会变成bss段的

#pragma data\_seg()

#pragma comment(linker,"/section:shareData,rws")

### 动态库调用

#### 隐式调用

需要lib文件，dll文件

声明函数

void show();

库文件使用声明

#pragma comment(lib,"kk.lib")

直接调用

show()

#### 显示调用

只需要dll文件

typedef void (\*pFun)();

HINSTANCE hInstanceLibrary=::LoadLibrary("kk.dll");

pFun fun=(pFun)GetProcAddress(hInstanceLibrary,"show");

fun();

::FreeLibrary(hInstanceLibrary);

### 调用类型

#### \_stdcall

Pascal程序的缺省调用方式，参数采用从右到左的压栈方式，被调函数自身在返回前清空堆栈。  
WIN32 Api都采用\_stdcall调用方式

#### \_cdecl

C/C++的缺省调用方式，参数采用从右到左的压栈方式，传送参数的内存栈由调用者维护。\_cedcl约定的函数只能被C/C++调用，每一个调用它的函数都包含清空堆栈的代码，所以产生的可执行文件大小会比调用\_stdcall函数的大。

由于\_cdecl调用方式的参数内存栈由调用者维护，所以变长参数的函数能使用这种调用约定

#### \_fastcall

调用较快，它通过CPU内部寄存器传递参数。

按C编译方式，\_fastcall调用约定在输出函数名前面加“@”符号，后面加“@”符号和参数的字节数，形如@functionname@number。

### 错误原因

头文件的作用是声明函数或者变量，如果缺少头文件，会出现找不到标识符的错误。

Lib文件的作用是指明头文件中声明的函数存在，lib文件包含实际代码的进入信息。如果没有引用lib，会出现unresolved external symbol（无法解析的外部符号）的错误。如果引用了lib，但是lib不在系统目录或者当前目录，会出现缺少lib的错误。(lib文件可以在连接器的附加依赖项中设置，也可以在代码中显式声明)

Dll文件存储着函数的实现代码，如果缺少dll，程序运行时会出错。

stl类型的参数无法作为引用或者指针进行传递。

### 调试

#### Dll工程调试

配置属性-》调试-》命令 输入调用exe所在位置

#### EXE调试

dll和pdb文件需要在一起

将dll的cpp文件直接在exe中打开即可

### 窗口交互

#### 创建模式对话框

函数前添加宏

AFX\_MANAGE\_STATE(AfxGetStaticModuleState());

#### 获取外部窗口

在库文件中需要调用外部窗口对象时，可以使用

HWND hWnd=GetForegroundWindow();

## SHELL

一般是指由操作系统提供的用于计算机用户向操作系统输入相关指令并得到结果的程序，shell可以字符形式的，也可以是GUI。Windows shell最重要的组成部件是explorer.exe，提供了开始菜单、任务栏、资源管理器等。

Shell API都是在shlobj.h文件中声明，由Shell32.dll导出，链接时需要Shell32.lib.

Shell扩展程序是COM(Component Object Model,组件对象模型)程序。COM程序一般是DLL文件（ActiveX程序是.ocx扩展名的文件）

### 遍历回收站

TCHAR pszPath[MAX\_PATH];

IShellFolder \*pisf=NULL;

IShellFolder \*pisfRecBin=NULL;

SHGetDesktopFolder(&pisfRecBin);

IEnumIDList \*peidl=NULL;

LPITEMIDLIST pidlBin=NULL;

LPITEMIDLIST idlCurrent=NULL;

SHGetSpecialFolderLocation(NULL,CSIDL\_BITBUCKET,&pidlBin);

pisfRecBin->BindToObject(pidlBin,NULL,IID\_IShellFolder,(void\*\*)&pisf);

pisf->EnumObjects(NULL,SHCONTF\_FOLDERS|SHCONTF\_NONFOLDERS|SHCONTF\_INCLUDEHIDDEN,&peidl);

STRRET strret;

ULONG uFetched;

while(1)

{

if(peidl->Next(1,&idlCurrent,&uFetched)==S\_FALSE)

break;

SHGetPathFromIDList(idlCurrent,pszPath);

pisf->GetDisplayNameOf(idlCurrent,SHGDN\_NORMAL,&strret);

CStringW str;

str.Format(L"%s",strret.pOleStr);

::MessageBoxW(m\_hWnd,str,L"xx",MB\_OK);

}

### CLSID

GUID(Global unique identier),全局唯一标识符，用于在系统中唯一标识一个对象。CLSID是GUID在注册表中的表示。

## function

### CString

#include <atlstr.h>

CString以”\0”结尾，如果字符串中在结尾前包含”\0”,那么CString以第一次出现”\0”为准。

LPCTSTR 指的就是CString，LPSTR指char \*

可用CString str(\_T("xxx"));构造

可用str.LoadString(ID)加载字符串资源中的字符串

str.Format(\_T("define %d %d"),wParam,lParam);

str.Empty(); 清空字符串

str+\_T(’c’); 为字符串增加一个字符

============= 追加

str.Append(\_T("x"));

str.AppendFormat(\_T("%d"),10);

=============查找

if(-1!=str.Find(\_T("clear")))//表示str中有clear字样,如果find返回-1代表没有}

=============分割

CString str = \_T("ab,cde");

int pos = str.Find(\_T(","));

str = str.Left(pos);//得到ab

str = str.Mid(pos+1);//得到cde,第二个参数如果不设置，代表从pos位置截取到末尾（加1因为查找的字符串为个长度）

或者：str = str.Right(str.GetLength()-pos-1);

=============修剪

CString str=\_T(" ab");

str.TrimLeft(); //可以将字符串左边多余的空格去掉

=============按条件截取

CString str=\_T("123abcd456");

CString s1=str.SpanIncluding(\_T("0123456789"));//s1为123

CString s2=str.SpanIncluding("a");//s2为空

CString s3=str.SpanExcluding(\_T("d"));//s3为123abc

=============比较

str1.Compare(str2); //如果str1与str2一样，返回0，否则返回1

str1.CompareNoCase(str2); //比较时忽略大小写

=============转换

char \*p=strA.GetBuffer(0);//CStringA中的GetBuffer就是转成char \*;GetBuffer后必须要调用ReleaseBuffer，否则CStringA的其他函数失效。

int i=\_ttoi(p); //字符串变成int

转成wchar\_t

int i=::MultiByteToWideChar(CP\_ACP,0,(LPCTSTR)file,-1,NULL,0);

wchar\_t \*pPath=new wchar\_t[i];

::MultiByteToWideChar(CP\_ACP,0,(LPCTSTR)file,-1,pPath,i);

delete[] pPath;

=========使用CString直接进行转换

CStringW strw = \_T("abc");

CStringA str(strw.GetBuffer());

string s(str.GetBuffer());

### 时间

SYSTEMTIME st;

GetLocalTime(&st); 得到系统时间，并放入到结构体中

CString str;

str.Format("%d:%d:%d",st.wHour,st.wMinute,st.wSecond);

计时：

LONGLONG t1 = GetTickCount64();

Sleep(60);

LONGLONG val = GetTickCount64() - t1;

### INI文件

通过设置段名和键名可以确定一个键值。可以进行写入和读取。

Ini文件写入情况如下（也可以用section的函数写没有键名的）

不能新建txt然后改成ini，第一次文件需要用函数写。文件名必须是全路径，不能是相对路径。

[Section1]

1=x

2=y

[Section2]

abc

::WriteProfileString等SDK函数可以在C:\WINDOWS\win.ini读写数据。如果是WriteProfileString等CWinApp的成员函数，则会写入注册表中。

#### 写入

::WritePrivateProfileString(strSectionName, strKeyName, strKeyValue, strFileName);

如果strKeyValue是NULL，则表示删除那一行

没有strKeyName时(debug下会有冗余数据，只有release可以)：

::WritePrivateProfileSection(strSectionName,strKeyValue,strFileName);

#### 读取

CString strKeyValue = \_T("");

::GetPrivateProfileString(strSectionName, strKeyName, \_T(""),

strKeyValue.GetBuffer(1024), 1024,strFileName);

strKeyValue.ReleaseBuffer();

没有strKeyName时：

::GetPrivateProfileSection(strSectionName,strKeyValue.GetBuffer(1024),1024,strFileName);

读取数字：（只用于带key的）

int x=GetPrivateProfileInt(strSectionName,strKeyValue,NULL,strFileName);

#### 遍历

void GetUnitINI(map<CString,CString> &Key\_Value,CString strSection,CString iniFileName)

{

while(true)

{

CString strKey;

DWORD nChar = GetPrivateProfileString(strSection,NULL,\_T(""),strKey.GetBuffer(1024),1024,iniFileName);

if (0 == nChar) break;

CString strValue;

GetPrivateProfileString(strSection,strKey,\_T(""),strValue.GetBuffer(1024),1024,iniFileName);

strValue.ReleaseBuffer();

Key\_Value[strKey] = strValue;

//删除

WritePrivateProfileString(strSection,strKey,NULL,iniFileName);

}

//重新写入

for (map<CString,CString>::const\_iterator iter = Key\_Value.begin(); iter != Key\_Value.end(); ++iter)

{

WritePrivateProfileString(strSection,iter->first,iter->second,iniFileName);

}

}

### CImage

#### 绘图及图像转换

#include <atlimage.h>

CImage Img;

if (S\_OK != Img.Load(L"D:\\1.jpg")) return;

HDC hdc = GetDC()->m\_hDC;

::SetStretchBltMode(hdc, COLORONCOLOR);//缩放绘制时必须设置，否则出现失真

Img.Draw(hdc, 0, 0, 100, 100);

int nchannels = img.GetBPP()/8;

Img.Save(L".\\abc.jpg");

#### 获取DIB

CImage Img;

Img.Attach(hDib);

Img.Draw(GetDC()->m\_hDC,0,0,Img.GetWidth(),Img.GetWidth());

可以代理hdib的销毁操作

### 调试

TRACE("xx");//可以将CString输出到调试窗口，debug有效

TRACE(\_T("\r\n"));

### 消息盒子

if(IDCANCEL== MessageBox(m\_hWnd,\_T("xx"),"标题",MB\_OK)) return;

### CMD

#include <io.h>

AllocConsole(); //FreeConsole();

HANDLE handle = GetStdHandle(STD\_OUTPUT\_HANDLE);

CString str = \_T("abc");

WriteConsole(handle, str.GetBuffer(), str.GetLength(), NULL, NULL);

========字体颜色

白底红字

system("color fc");

FOREGROUND\_RED

SetConsoleTextAttribute(GetStdHandle(STD\_OUTPUT\_HANDLE),0xfc);

//0xfa 绿色字白色底

### 获取命令行参数

int num = \_\_argc;

TCHAR \*\*args = \_\_targv;

### 特殊目录获取

#include <ShlObj.h>

CString GetSpecialFolderPath(UINT CSIDL)

{

TCHAR szMyDocument[MAX\_PATH] = {0};

SHGetSpecialFolderPath(NULL,szMyDocument,CSIDL,FALSE);

return szMyDocument;

}

CSIDL\_SYSTEM 系统dll放置位置 GetSystemDirectory

CSIDL\_SYSTEMX86 32位dll放置位置

CSIDL\_DESKTOP 桌面

CSIDL\_STARTUP 启动目录

CSIDL\_COMMON\_STARTMENU 开始菜单（可以创建文件夹）

CSIDL\_FONTS 字体目录

### 创建快捷方式

#include <ShlObj.h>

void CreateShortCut(CString filePathName,CString destPath,CString destName,CString description = \_T(""))

{//destPath后面不带结束符

if (!PathFileExists(filePathName)) throw filePathName + \_T(" 不存在");

CString destPathName = destPath + \_T("\\") + destName + \_T(".lnk");

IShellLink \*plink = NULL;

HRESULT hr = CoCreateInstance(CLSID\_ShellLink,NULL,CLSCTX\_INPROC\_SERVER,IID\_IShellLink,(void\*\*)&plink);

if(FAILED(hr)) throw CString(\_T("创建快捷方式实例错误"));

IPersistFile \*ppf = NULL;

hr = plink->QueryInterface(IID\_IPersistFile,(void\*\*)&ppf);

if (FAILED(hr))

{

plink->Release();

throw CString(\_T("创建快捷方式QueryInterface错误"));

}

plink->SetPath(filePathName);

plink->SetDescription(description);

plink->SetShowCmd(SW\_NORMAL);

hr = ppf->Save(destPathName,TRUE);

ppf->Release();

plink->Release();

if(!SUCCEEDED(hr)) CString(\_T("创建快捷方式失败"));

}

### 获取窗口位置

RECT rect;

GetClientRect(hWnd,&rect);得到客户区坐标（大小），相对于父窗口的坐标

GetWindowRect(hWnd,&rect);得到当前窗口的坐标（大小），相对于屏幕坐标系的坐标

### 窗口大小和位置

int cx=::GetSystemMetrics(SM\_CXSCREEN);

int cy=::GetSystemMetrics(SM\_CYSCREEN);

如果是窗口，以屏幕坐标系为基准（窗口属于屏幕）。如果是窗口内部控件，就以窗口为基准（控件属于窗口）。

SetWindowPos(hWnd,NULL, x, y, cx, cy,SWP\_SHOWWINDOW);

第一个参数指z序对应的父窗口

如果最后一个参数是SWP\_NOZORDER时，x，y同时为0，则该函数只能设置大小，位置会自动调节到屏幕中间。SWP\_NOMOVE可以使窗口无法移动。

MoveWindow(hWnd, x, y, cx, cy,TRUE);

主窗口使用MoveWindow时如果x，y同时为0，则会自动移到中间。

### 顶层窗口

SetWindowPos(hWnd, HWND\_TOPMOST, x, y, cx, cy, SWP\_SHOWWINDOW);

### 鼠标

if(GetAsyncKeyState(VK\_LBUTTON)==0) //判断全局的鼠标左键是否弹起

ShowCursor(FALSE); //显示或隐藏鼠标

POINT pt;

GetCursorPos(&pt);//得到当前鼠标位置，相对于屏幕坐标系。

或者：

DWORD dwPos = GetMessagePos();

POINT pt = { LOWORD(dwPos), HIWORD(dwPos) };

ScreenToClient(hWnd,&pt);//转换为客户区坐标系

//判断鼠标是否在某个区域

PtInRect(rect,pt)

### 获取鼠标滚动量

case WM\_MOUSEWHEEL:

{

POINT pt;

GetCursorPos(&pt);

ScreenToClient(hWnd,&pt);//获取当前鼠标在对话框中的位置

int delta = GET\_WHEEL\_DELTA\_WPARAM(wParam);//获取滚动量

}

SetFocus(hWnd);//按钮点击时会改变鼠标焦点，此时需要将焦点返回到窗口，否则滚轮消息收不到

### 模拟鼠标消息

SetCursorPos(x,y);

mouse\_event(MOUSEEVENTF\_LEFTDOWN,0,0,0,0);

mouse\_event(MOUSEEVENTF\_LEFTUP,0,0,0,0);

### 播放声音

WAV格式

#include <Mmsystem.h>

#pragma comment(lib, "winmm.lib")

==============播放文件

sndPlaySound(\_T("D:\\2.wav"),SND\_ASYNC);

==============播放资源文件

PlaySound(MAKEINTRESOURCE(IDR\_WAVE1),AfxGetResourceHandle(),SND\_ASYNC|SND\_RESOURCE|SND\_NODEFAULT|SND\_LOOP);

==============加载资源文件播放

HMODULE hmod=AfxGetResourceHandle();

HRSRC hSndResource=FindResource(hmod,MAKEINTRESOURCE(IDR\_WAVE1),\_T("WAVE"));

HGLOBAL m\_hGlobalMem=LoadResource(hmod,hSndResource);

LPCTSTR m\_lpMemSound=(LPCSTR)LockResource(m\_hGlobalMem);

sndPlaySound(m\_lpMemSound,SND\_MEMORY);

//FreeResource(m\_hGlobalMem);

### 后台运行

WINDOWPLACEMENT wpOrg;

GetWindowPlacement(hWnd,&wpOrg);//恢复时用

WINDOWPLACEMENT wp;

wp.length = sizeof(WINDOWPLACEMENT);

wp.flags = WPF\_RESTORETOMAXIMIZED;

wp.showCmd = SW\_HIDE;

SetWindowPlacement(hWnd,&wp);

### 只运行一个程序实例

const CString g\_szPropName =\_T("Your Prop Name");

const HANDLE g\_hValue = (HANDLE)1;

BOOL CALLBACK EnumWndProc(HWND hwnd,LPARAM lParam)

{

HANDLE h = GetProp(hwnd,g\_szPropName);

if( h == g\_hValue)

{

\*(HWND\*)lParam = hwnd;

return false;

}

return true;

}

======程序开始运行时判断：

HWND oldHWnd = NULL;

EnumWindows(EnumWndProc,(LPARAM)&oldHWnd); if(oldHWnd != NULL)

{ ::ShowWindow(oldHWnd,SW\_SHOW);

::SetForegroundWindow(oldHWnd);

return false;

}

======程序启动后，设置程序标志：

BOOL addNewHandle=SetProp(m\_hWnd,g\_szPropName,g\_hValue);

//设置当前程序的标识，使下次运行时可以作为判断依据

======释放标记

RemoveProp(m\_hWnd,g\_szPropName);

### 内存监控

MEMORYSTATUSEX mems;

mems.dwLength = sizeof(mems);

GlobalMemoryStatusEx(&mems);

DWORDLONG ullTotalPhys = mems.ullTotalPhys/1024/1024;//计算机内存

DWORDLONG ullAvailPhys = mems.ullAvailPhys/1024/1024;//计算机可用内存 8g内存32位系统可以用4g

DWORDLONG ullTotalVirtual = mems.ullTotalVirtual/1024/1024;//m

DWORDLONG ullAvailVirtual = mems.ullAvailVirtual/1024/1024;//m

DWORDLONG usedMem = ullTotalVirtual - ullAvailVirtual; //当前已经使用的内存

### 窗口去色

void SetAlpha(HWND wnd, COLORREF alphaClr = RGB(0, 0, 0))

{

#define LWA\_COLORKEY 0x00000001

#define LWA\_ALPHA 0x00000002

#define WS\_EX\_LAYERED 0x00080000

typedef BOOL(WINAPI \*LPFNSETLAYEREDWINDOWATTRIBUTES)(HWND hWnd, COLORREF crKey, BYTE bAlpha, DWORD dwFlags);

LPFNSETLAYEREDWINDOWATTRIBUTES SetLayeredWindowAttributes;

HMODULE hUser32 = GetModuleHandle(\_T("user32.dll"));

SetLayeredWindowAttributes = (LPFNSETLAYEREDWINDOWATTRIBUTES)GetProcAddress(hUser32, "SetLayeredWindowAttributes");

SetWindowLong(wnd, GWL\_EXSTYLE, GetWindowLong(wnd, GWL\_EXSTYLE) | WS\_EX\_LAYERED);

SetLayeredWindowAttributes(wnd, alphaClr, 0, LWA\_COLORKEY);

}

### 定时器

SetTimer(hWnd, 0, 1000, NULL);

第二个参数为定时器id，可任意，第三个1000代表1秒，第四个代表使用hWnd中的处理函数,如果为NULL，则使用WM\_TIMER（wParam 代表id）

### 显示或隐藏任务栏

//隐藏任务栏

HWND wnd = FindWindow(\_T("Shell\_TrayWnd"), NULL);

if (IsWindowVisible(wnd)) ShowWindow(wnd,SW\_HIDE);

//显示任务栏

HWND wnd = FindWindow(\_T("Shell\_TrayWnd"), NULL);

if (!IsWindowVisible(wnd)) ShowWindow(wnd, SW\_SHOW);

### 注销、关闭、重启系统

#### 使用前相关操作

if (IDNO == MessageBox(\_T("是否进行\*\*操作？"), \_T("对话框标题"), MB\_YESNO)){

return;

}

//打开进程令牌

HANDLE hToken;

if (!OpenProcessToken(GetCurrentProcess(),

TOKEN\_ADJUST\_PRIVILEGES | TOKEN\_QUERY, &hToken))

{

AfxMessageBox(\_T("OpenProcessToken Error."));

}

//获得LUID

TOKEN\_PRIVILEGES tkp;

LookupPrivilegeValue(NULL, SE\_SHUTDOWN\_NAME, &tkp.Privileges[0].Luid);

tkp.PrivilegeCount = 1;

tkp.Privileges[0].Attributes = SE\_PRIVILEGE\_ENABLED;

//调整令牌权限

AdjustTokenPrivileges(hToken, FALSE, &tkp, 0, (PTOKEN\_PRIVILEGES)NULL, 0);

if (GetLastError() != ERROR\_SUCCESS) return;

#### 注销

if (!ExitWindowsEx(EWX\_LOGOFF, 0)) return;

#### 关闭

if (!ExitWindowsEx(EWX\_SHUTDOWN, 0)) return;

#### 重启

if (!ExitWindowsEx(EWX\_REBOOT, 0)) return;

### 窗口标题操作

SetWindowText(hWnd,\_T("窗口标题"));

CString rString;

GetWindowText(hWnd, rString.GetBuffer(1024),1024);

### 打开文件夹对话框

inline bool GetDirPath(CString &strFolderPath,HWND wndParent=NULL,CString Title = \_T("请选择一个文件夹"))

{

TCHAR szFolderPath[MAX\_PATH] = {0};

BROWSEINFO sInfo;

::ZeroMemory(&sInfo, sizeof(BROWSEINFO));

sInfo.pidlRoot = 0;

sInfo.lpszTitle = Title;

sInfo.ulFlags = BIF\_DONTGOBELOWDOMAIN | BIF\_RETURNONLYFSDIRS | BIF\_NEWDIALOGSTYLE | BIF\_EDITBOX;

sInfo.lpfn = NULL;

sInfo.hwndOwner = wndParent;

// 显示文件夹选择对话框

LPITEMIDLIST lpidlBrowse = ::SHBrowseForFolder(&sInfo);

if (lpidlBrowse != NULL)

{

// 取得文件夹名

if (::SHGetPathFromIDList(lpidlBrowse,szFolderPath))

{

strFolderPath = szFolderPath;

}

::CoTaskMemFree(lpidlBrowse);

}

if(lpidlBrowse == NULL) return false;

return true;

}

### 打开文件对话框

#include "commdlg.h"

#pragma comment(lib,"Comdlg32.lib")

bool ChoiceFile(CString &filePathName,const TCHAR\* filter)

{

//\_T("xx(\*.jpg)\0\*.jpg\0所有文件（\*.\*）\0\*.\*\0");

TCHAR szFileName[MAX\_PATH] = {};

OPENFILENAMEW openFileName = {};

openFileName.lStructSize = sizeof(OPENFILENAMEW);

openFileName.nMaxFile = MAX\_PATH;//这个必须设置，不设置的话不会出现打开文件对话框

openFileName.lpstrFilter = filter;

openFileName.lpstrFile = szFileName;

openFileName.nFilterIndex = 1;

openFileName.Flags = OFN\_PATHMUSTEXIST | OFN\_FILEMUSTEXIST;

// 如果保存则用GetSaveFileName()

if (!GetOpenFileName(&openFileName)) return false;

filePathName = openFileName.lpstrFile;

return true;

}

### 读写注册表

//打开注册表 用户环境变量

HKEY hKey;

if(RegOpenKey(HKEY\_CURRENT\_USER, \_T("Environment"), &hKey) != ERROR\_SUCCESS) return;

//读取 path环境变量

LPBYTE data[2048] = {0};

DWORD lValueSize;

if (RegQueryValueExA(hKey,"path",NULL,NULL,(LPBYTE)data,&lValueSize) == ERROR\_SUCCESS)

{

CStringA re;

re.Format("%s",data);

}

//设置环境变量

if(RegSetValueExA(hKey,strName.c\_str(),NULL,regType,(BYTE\*)strValue.c\_str(),strValue.length()) != ERROR\_SUCCESS) return;

### 开机启动

bool SetStartInfo(TCHAR \*KeyName,bool remove = false)

{

HKEY hKey;

TCHAR\* lpRun = \_T("Software\\Microsoft\\Windows\\CurrentVersion\\Run");

long lRet = RegOpenKeyEx(HKEY\_CURRENT\_USER, lpRun, 0, KEY\_WRITE, &hKey);

if(ERROR\_SUCCESS != lRet) return false;

TCHAR pFileName[MAX\_PATH] = {0};

DWORD dwRet = GetModuleFileName(NULL, pFileName, MAX\_PATH);

DWORD strSize = dwRet\*sizeof(TCHAR);

if (remove)

{//删除

lRet = RegDeleteValue(hKey,KeyName);

}

else

{//添加

lRet = RegSetValueEx(hKey, KeyName, 0, REG\_SZ, (BYTE \*)pFileName, strSize);

}

RegCloseKey(hKey);

if(lRet != ERROR\_SUCCESS) return false;

return true;

}

### 注册ActiveX 控件

BOOL DllRegister(LPCTSTR lpszDllName)

{

ASSERT(lpszDllName != NULL);

ASSERT(AfxIsValidString(lpszDllName));

//加载ActiveX控件

HINSTANCE hLib = LoadLibrary(lpszDllName);

if (hLib == NULL)

{

TRACE(\_T("%s加载失败\n"), lpszDllName);

return FALSE;

}

//获得注册函数DllRegisterServer地址

FARPROC lpDllEntryPoint;

lpDllEntryPoint = GetProcAddress(hLib, \_T("DllRegisterServer"));

//调用注册函数DllRegisterServer

if (lpDllEntryPoint != NULL)

{

if (FAILED((\*lpDllEntryPoint)()))

{

TRACE(\_T("调用DllRegisterServer失败\n"));

FreeLibrary(hLib);

return FALSE;

}

else

{

FreeLibrary(hLib);

return TRUE;

}

}

else

{

TRACE(\_T("调用DllRegisterServer失败\n"));

FreeLibrary(hLib);

return FALSE;

}

}

### 模拟键盘按键

#### 模拟NUM LOCK按键

keybd\_event(VK\_NUMLOCK, 0, KEYEVENTF\_EXTENDEDKEY | 0, 0);

keybd\_event(VK\_NUMLOCK, 0, KEYEVENTF\_EXTENDEDKEY | KEYEVENTF\_KEYUP, 0);

#### 模拟CAPS LOCK按键

keybd\_event(VK\_CAPITAL, 0, KEYEVENTF\_EXTENDEDKEY | 0, 0);

keybd\_event(VK\_CAPITAL, 0, KEYEVENTF\_EXTENDEDKEY | KEYEVENTF\_KEYUP, 0);

#### 模拟SCROLL LOCK按键

keybd\_event(VK\_SCROLL, 0, KEYEVENTF\_EXTENDEDKEY | 0, 0);

keybd\_event(VK\_SCROLL, 0, KEYEVENTF\_EXTENDEDKEY | KEYEVENTF\_KEYUP, 0);

## file

### 基本概念

#### 磁盘分区

一个物理硬盘上划分的可以独立工作的一些逻辑磁盘。

#### 卷

也称为逻辑驱动器，是ntfs等文件系统的最高层。是存储设备（硬盘）上文件系统管理的一块区域，在逻辑上互相隔离的存储单元。一个磁盘分区至少含有一个卷。仅存在于一个分区上的卷称为“简单卷”，仅存在与多个分区上的卷称为“多分区卷”或“跨区卷”。一般情况下，一个分区只包含一个卷，一个卷也只存在于一个分区上。卷存在卷标，程序可以通过卷标访问卷。

### 获取硬盘信息

DWORD VolumeSerialNumber;

GetVolumeInformation("C:\\",NULL,12,&VolumeSerialNumber,NULL,NULL,NULL,10);

//CString ss;

//ss.Format("%x",VolumeSerialNumber);

可以获取c盘的序列号，如果第一个参数不指定位置，则获取的是当前位置的号码

#### 遍历卷

DWORD len=1024;

CHAR ch[1024];

ZeroMemory(ch,1024);

GetLogicalDriveStrings(len,ch);

CHAR\* p=ch;

while(\*p!='\0')

{

CString str;

str.Format("%s",p);

AfxMessageBox(str);//得到所有的卷标

p+=lstrlen(p)+1;

}

#### 读文件

HANDLE hFile;

hFile=CreateFile("d:\\a.txt",GENERIC\_READ,0,NULL,OPEN\_EXISTING,FILE\_ATTRIBUTE\_NORMAL,NULL);

char ch[100];

DWORD dwReads;

ReadFile(hFile,ch,100,&dwReads,NULL); //读取文件放到ch中

ch[dwReads]=0;

CloseHandle(hFile);

#### 写文件

HANDLE hFile;

hFile=CreateFile("d:\\a.txt",GENERIC\_WRITE,0,NULL,CREATE\_NEW,FILE\_ATTRIBUTE\_NORMAL,NULL);

DWORD dwWrites;

WriteFile(hFile,"abcde",strlen("abcde"),&dwWrites,NULL);

CloseHandle(hFile);

### 目录操作

#include <shlwapi.h>

#### 获取当前目录

VS中，当前目录指工程所在目录

一般直接写”.\\”

有时不能使用相对路径，只能用绝对路径

GetCurrentDirectory(1024,str.GetBuffer(1024));

//获取exe所在目录

CString GetCurPath()

{

CString path;

GetModuleFileName(NULL, path.GetBuffer(MAX\_PATH), MAX\_PATH);

path.ReleaseBuffer();

path = path.Left(path.ReverseFind(\_T('\\')));

return path;

}

#### 获取模块名称

第一个参数为null时，获取当前模块（exe）的全路径（包括文件名）

GetModuleFileName(NULL,str.GetBuffer(1024),1024);

#### 设置进程当前目录

SetCurrentDirectory("d:\\");

设置后，程序中使用.\\即为d:\\

#### 判断是否存在

也可以用于判断网络目录是否联通

CString strPath=\_T("d:\\test");

if(PathIsDirectory(strPath)) { //存在}

else{ //不存在}

#### 创建目录

#include <shlobj.h>

(void)SHCreateDirectoryEx(NULL,"D:\\abc\\efg",NULL);

### 文件查找

void DirList(CString Path,vector<CString> &fileVec)

{

WIN32\_FIND\_DATA FindData;

HANDLE hError = FindFirstFile(Path + \_T("\\\*.\*"), &FindData);

if (hError == INVALID\_HANDLE\_VALUE) return;

CString DotDir = \_T(".");

CString TowDotDir = \_T("..");

while (::FindNextFile(hError, &FindData))

{

if (DotDir == FindData.cFileName || TowDotDir == FindData.cFileName) continue;

CString FullPathName = Path + \_T("\\") + FindData.cFileName;

if (FindData.dwFileAttributes & FILE\_ATTRIBUTE\_DIRECTORY)

{

DirList(FullPathName, fileVec);

}

else

{

fileVec.push\_back(FullPathName);

}

}

}

### 删除

#### 删除文件

DeleteFile("d:\\1.txt");

删除后回收站无信息

#### 删除目录

只能用于非空目录删除

RemoveDirectory(directory\_path);

### 文件操作

#### 文件是否存在

PathFileExists(strPath)；

#### 复制

CopyFile("d:\\1.jpg","d:\\11.jpg",TRUE);

将1.jpg复制为11.jpg，如果已经存在，则覆盖（TRUE指定）

#### 移动/重命名

可以移动文件或者目录

MoveFile("d:\\1.jpg","d:\\ATY\\1.jpg");

#### 文件另存为

CopyFile(\_T(".\\a.txt"),\_T("d:\\a.txt"),FALSE);//将当前目录下的a.txt保存到d盘

#### 内存方式读写文件

通过使用Mapping File对文件进行如同读写内存方式的操作

#define FILE\_MAP\_START 0x28904

#define BUFFSIZE 1024

SYSTEM\_INFO SysInfo;

::GetSystemInfo(&SysInfo);

DWORD dwSysGran=SysInfo.dwAllocationGranularity;//获取系统内存分配粒度

DWORD dwMapViewSize=(FILE\_MAP\_START%dwSysGran)+BUFFSIZE;

DWORD dwFileMapSize=FILE\_MAP\_START+BUFFSIZE;

HANDLE hFile=CreateFile("d:\\2.jpg",GENERIC\_READ|GENERIC\_WRITE,FILE\_SHARE\_READ|FILE\_SHARE\_WRITE,NULL,OPEN\_EXISTING,FILE\_ATTRIBUTE\_NORMAL,NULL);

HANDLE hMapFile=CreateFileMapping(hFile,NULL,PAGE\_READWRITE,0,dwFileMapSize,NULL);

if(hMapFile==NULL)

{

return ;

}

LPVOID lpMapAddress=MapViewOfFile(hMapFile,FILE\_MAP\_ALL\_ACCESS,0,

0,dwMapViewSize);//代表从起始位置开始

if(lpMapAddress==NULL)

{

return ;

}

char\* pData=(char \*)lpMapAddress;//char\*指针（也可以是int\*等指针），相当于直接指向文件，此时对通过该指针可以对文件进行类似指针的操作

\*pData=\*pData^22; //可以对指定位置进行两次异或操作，即变化后再还原

FlushViewOfFile(lpMapAddress,dwMapViewSize);//操作文件视图后，写入文件

CloseHandle(hFile);

#### 中文支持

#include <locale>

setlocale(LC\_CTYPE,"chs");

unicode下无法输出中文，需要设置

## IPC

### 获取PID

通过遍历操作系统进程获取进程名称和PID

#include <tlhelp32.h>

bool getPid(DWORD &findPID, CString findExeName)

{//findExeName名称中包含.exe

bool hasfind = false;

PROCESSENTRY32 pe32;

pe32.dwSize = sizeof(pe32);

HANDLE hProcessSnap = ::CreateToolhelp32Snapshot(TH32CS\_SNAPPROCESS, 0);

if (hProcessSnap == INVALID\_HANDLE\_VALUE)

{

MessageBox(NULL,TEXT("CreateToolhelp32Snapshot调用失败!"),TEXT(""),MB\_OK);

return hasfind;

}

BOOL bMore = ::Process32First(hProcessSnap, &pe32);

while (bMore)

{

CString exeName = pe32.szExeFile;

DWORD pid = pe32.th32ProcessID;

if (exeName == findExeName)

{

findPID = pid;

hasfind = true;

break;

}

bMore = ::Process32Next(hProcessSnap, &pe32);

}

::CloseHandle(hProcessSnap);

return hasfind;

}

### 获取hProcess

HWND wnd = ::FindWindow(NULL,wndName);

if (NULL == wnd) return;

DWORD pid;

GetWindowThreadProcessId(wnd,&pid);

HANDLE hProcess = OpenProcess(PROCESS\_ALL\_ACCESS,FALSE,pid);

### DLL注入

void InjectHookDLL(HANDLE hProcess,PCWSTR dllFilePathName)

{

int cch = 1 + lstrlenW(dllFilePathName);

int cb = cch \* sizeof(wchar\_t);

PWSTR PszLibFileRemote = (PWSTR)VirtualAllocEx(hProcess, NULL, cb, MEM\_COMMIT, PAGE\_READWRITE);

WriteProcessMemory(hProcess, PszLibFileRemote, (PVOID)dllFilePathName, cb, NULL);

HANDLE hThread = CreateRemoteThread(hProcess, NULL, 0, (PTHREAD\_START\_ROUTINE)LoadLibrary, PszLibFileRemote, 0, NULL);

WaitForSingleObject(hThread, INFINITE);

}

### 钩子

#### 进程内钩子

::SetWindowsHookEx(WH\_MOUSE,MouseProc, NULL, GetCurrentThreadId());

#### 监听程序消息

动态库为所有被监听对象共享（每个被监听的程序触发监听事件后都会加载该动态库）

CreatorPID为0，则监听所有进程

##### 动态库

LRESULT CALLBACK KeyboardProc(int code, WPARAM wParam, LPARAM lParam)

{

\*\*\*\*\*\*\*\*

return 0;//返回0，代表将消息传出去，[如果是return 1，则拦截消息]

}

extern "C" \_declspec(dllexport) void Run(DWORD CreatorPID,const char \*dllName)

{

SetWindowsHookEx(WH\_KEYBOARD, KeyboardProc, ::GetModuleHandle(CString(dllName).GetBuffer()), CreatorPID);

}

##### 使用程序激活动态库

HWND hWnd = ::FindWindow(0, \_T("MFCApplication2"));

if (NULL == hWnd)return;

DWORD CreatorPID = GetWindowThreadProcessId(hWnd, 0);

HMODULE h = LoadLibrary(\_T("\*\*\*\*\*\\Release\\hh.dll"));

typedef void (\*pfun)(DWORD CreatorPID, const char \*dllName);

pfun fun = (pfun)GetProcAddress(h, "Run");

fun(CreatorPID,"hh.dll");

#### 卸载钩子

HHOOK hHook=SetWindowsHookEx(\*\*\*\*\*\*\*);

UnhookWindowsHookEx(hHook);

#### 钩子传递

钩子处理函数中调用

CallNextHookEx(0, nCode, wParam, lParam);

//将消息传递给钩子链中的下一个钩子函数

#### 键盘钩子

WH\_KEYBOARD

LRESULT CALLBACK KeyboardProc(

int code, // 按键消息状态

WPARAM wParam, // 虚键码（VK\_ESCAPE这种），对应键盘ASCII码

LPARAM lParam // 按键信息

)

##### code

HC\_ACTION代表lParam中包含按键信息

HC\_NOREMOVE代表lParam中包含按键信息，但是消息没有从队列取出，比如程序调用的是PeakMessage。

##### wParam

//获取键盘按键a~z，并放入str中(使用ANSI)

if(wParam>=65&&wParam<=90)

{

TCHAR ch=wParam;

CString str;

str.Format(\_T("%c"),ch);

}

##### lParam

31位标识了按键是处于按下还是弹起，1代表弹起。

处理键盘消息前需要判断，否则会出现两次按键消息。

if(lParam>>31&1)

0-15位标识按键重复的次数

29位标识alt键是否按下，1代表按下

//屏蔽Alt+F4组合键

if(VK\_F4==wParam && (lParam)>>29&1)

return 1;

if(((DWORD)lParam&0x40000000) && (HC\_ACTION==code))

{

BOOL b\_sft=::GetAsyncKeyState (VK\_SHIFT)>>((sizeof(short)\*8)-1);

//代表是否同时按下shift键

}

#### 鼠标钩子

WH\_MOUSE

鼠标钩子的回调函数MouseProc中一定要对code,wParam的状态进行判断，否则因为鼠标的状态一直变化，执行语句会变成死循环。

if (0 != code) return 0;

WM\_MOUSEMOVE==wParam 表示鼠标移动消息

LRESULT CALLBACK MouseProc(

int nCode, // hook code

WPARAM wParam, // message identifier

LPARAM lParam // mouse coordinates

)

在MouseProc中可以获取位置信息

PMOUSEHOOKSTRUCT pStruct=(PMOUSEHOOKSTRUCT)lParam;

WPARAM x = pStruct->pt.x;

WPARAM y = pStruct->pt.y;

LPARAM 是指针，所以不能用postMessage传递

### 打开其他程序

#### WinExec

WinExec("\*\*\*\*\*.exe",SW\_HIDE);

#### CreateProcess

PROCESS\_INFORMATION processInfo;

STARTUPINFO startupInfo;

::ZeroMemory(&startupInfo, sizeof(startupInfo));

startupInfo.cb = sizeof(startupInfo);

startupInfo.wShowWindow = FALSE;

if (!::CreateProcess(\_T("E:\\bbb.exe"), NULL,

NULL, // process security

NULL, // thread security

FALSE, // no inheritance

0, // no startup flags

NULL, // no special environment

NULL, // default startup directory

&startupInfo,

&processInfo)) return;

包含程序的句柄和PID

processInfo.hProcess

processInfo.dwProcessId

#### ShellExecuteEx

======打开并等待结束

文件路径和参数字符串必须是字面值常量

SHELLEXECUTEINFO ShExecInfo = { 0 };

ShExecInfo.cbSize = sizeof(SHELLEXECUTEINFO);

ShExecInfo.fMask = SEE\_MASK\_NOCLOSEPROCESS;

ShExecInfo.hwnd = NULL;

ShExecInfo.lpVerb = NULL;

ShExecInfo.lpFile = \_T("C:\\Python36\\python.exe");//exe或文件

ShExecInfo.lpParameters = \_T("D:\\t.py");//参数

ShExecInfo.lpDirectory = NULL;

ShExecInfo.nShow = SW\_HIDE;//后台打开程序

ShExecInfo.hInstApp = NULL;

ShellExecuteEx(&ShExecInfo);

WaitForSingleObject(ShExecInfo.hProcess, INFINITE);//等待结束

CloseHandle(ShExecInfo.hProcess);

======结束程序

TerminateProcess(sei.hProcess,0);

#### 控制其他程序

使用VS下 工具->SPY++ 选择工具查找窗口，可找到对应程序窗口的类名和窗口标题

HWND hWnd=::FindWindow(NULL,\_T("name"));

if(hWnd==NULL){return;}

PostMessage (hWnd,WM\_USER+1,0,0);

//只能接收PostMessage发的消息

如果接收消息的程序的border属性为none，则需要初始时：

SetWindowText(hWnd, TEXT("name"));

移动目标程序的窗口

::MoveWindow(hWnd,0,0,500,500,TRUE);

### 读写目标程序内存

void WriteExeMemory(HANDLE hProcess,UINT64 addr,DWORD val)

{

//DWORD readVal = 0;

//ReadProcessMemory(hProcess,(LPVOID)addr,&readVal,sizeof(readVal),0);

BOOL ret = WriteProcessMemory(hProcess,(LPVOID)addr,&val,sizeof(val),0);

}

### WM\_COPYDATA

发送WM\_COPYDATA后，要等客户端执行该消息函数结束后

::SendMessage(hWnd,(UINT)WM\_COPYDATA,0,(LPARAM)&cds);代码之后的才执行

WM\_COPYDATA传输只读数据

#### 发送端

HWND hWnd=::FindWindow(NULL,\_T("name")); //接收端窗口名

if(hWnd==NULL){return;}

COPYDATASTRUCT cds;

cds.dwData=1;//用户定义数据

cds.lpData="abcd";//数据指针

cds.cbData=5;//数据长度

::SendMessage(hWnd,(UINT)WM\_COPYDATA,(WPARAM)m\_hWnd,(LPARAM)&cds);

#### 接收端

处理WM\_COPYDATA消息

BOOL C\*\*Dlg::OnCopyData(CWnd\* pWnd, COPYDATASTRUCT\* pCopyDataStruct)

{

CStringA str = (LPSTR)pCopyDataStruct->lpData;

return CDialog::OnCopyData(pWnd, pCopyDataStruct);

}

//PCOPYDATASTRUCT lpcds=(PCOPYDATASTRUCT)lParam;

//(LPSTR)lpcds->lpData

### 共享内存

#define Share\_Name \_T("sharename")

struct Stu

{

int acc;

char ch[100];

};

const int ShareSize = sizeof(Stu);

#### 创建端

Stu\* Create()

{

HANDLE hMapFile =CreateFileMapping(INVALID\_HANDLE\_VALUE,NULL,PAGE\_READWRITE,0,ShareSize,Share\_Name);

if(hMapFile==NULL||hMapFile==INVALID\_HANDLE\_VALUE) return NULL;

return (Stu\*)MapViewOfFile(hMapFile,FILE\_MAP\_ALL\_ACCESS,0,0,ShareSize);

}

#### 读取端

Stu\* Read()

{

HANDLE hMapFile=OpenFileMapping(FILE\_MAP\_ALL\_ACCESS,FALSE,Share\_Name);

if(hMapFile==NULL) return NULL;

Stu \*ptr = (Stu\*)MapViewOfFile(hMapFile,FILE\_MAP\_ALL\_ACCESS,0,0,ShareSize);

CloseHandle(hMapFile);

return ptr;

}

### 剪贴板

#### 写入

void WriteClipboard(char\* pStr,int strLen)

{

if (!OpenClipboard(NULL)) return;

EmptyClipboard();

HANDLE hClip = GlobalAlloc(GMEM\_MOVEABLE, strLen);

void \*pBuf = GlobalLock(hClip);

memcpy\_s(pBuf, strLen, pStr, strLen);

SetClipboardData(CF\_TEXT, hClip);//CF\_TEXT表示当前使用文本类型数据

GlobalUnlock(hClip);

CloseClipboard();

}

#### 获取

CString GetClipboard()

{

if (!OpenClipboard(NULL)) return \_T("");

if (!IsClipboardFormatAvailable(CF\_TEXT)) return \_T("");

HANDLE hClip = GetClipboardData(CF\_TEXT);

if (NULL == hClip) return \_T("");

char \*pBuf = (char\*)GlobalLock(hClip); //取出数据

CString str(pBuf);

GlobalUnlock(hClip);

CloseClipboard();

return str;

}

### 有名管道

Windows上的客户端-服务器模式

#### 服务端

HANDLE hPipe=NULL;

析构时：

if(hPipe){CloseHandle(hPipe);}

初始化：

hPipe=CreateNamedPipe(\_T("\\\\.\\pipe\\Mypipename"),PIPE\_ACCESS\_DUPLEX|FILE\_FLAG\_OVERLAPPED,0,1,1024,1024,0,NULL); //第一个参数格式固定，只有最后的名字Mypipename可以改变

if(INVALID\_HANDLE\_VALUE==hPipe)

{

hPipe=NULL;

return ;

}

HANDLE hEvent=CreateEvent(NULL,TRUE,FALSE,NULL);

if(!hEvent)

{

CloseHandle(hPipe);

hPipe=NULL;

return ;

}

OVERLAPPED overlap;

ZeroMemory(&overlap,sizeof(OVERLAPPED));

overlap.hEvent=hEvent;

if(!ConnectNamedPipe(hPipe,&overlap))//等待客户端连接

{

if(ERROR\_IO\_PENDING!=GetLastError())

{

CloseHandle(hPipe);

CloseHandle(hEvent);

hPipe=NULL;

return ;

}

}

if(WAIT\_FAILED==WaitForSingleObject(hEvent,INFINITE))

{

CloseHandle(hPipe);

CloseHandle(hEvent);

hPipe=NULL;

return ;

}

CloseHandle(hEvent);

读取：

TCHAR buf[100];

DWORD dwRead;

if(!ReadFile(hPipe,buf,100,&dwRead,NULL))

{return ;}

CString str;

str.Format(\_T("%s"),buf);

写入：

TCHAR buf[]=\_T("服务端......"); //要写入的数据

DWORD dwWrite;

if(!WriteFile(hPipe,buf,lstrlen(buf)+1,&dwWrite,NULL))

{return ;}

#### 客户端

HANDLE hPipe=NULL;

析构时：

if(hPipe){CloseHandle(hPipe);}

初始化：

if(!WaitNamedPipe(\_T("\\\\.\\pipe\\Mypipename"),NMPWAIT\_WAIT\_FOREVER))

{return ;}

hPipe=CreateFile(\_T("\\\\.\\pipe\\Mypipename"),GENERIC\_READ|GENERIC\_WRITE,0,NULL,OPEN\_EXISTING,FILE\_ATTRIBUTE\_NORMAL,NULL);

if(INVALID\_HANDLE\_VALUE==hPipe)

{

hPipe=NULL;

return ;

}

读取：

TCHAR buf[100];

DWORD dwRead;

if(!ReadFile(hPipe,buf,100,&dwRead,NULL))

{return ;}

CString str;

str.Format(\_T("%s"),buf);

写入：

TCHAR buf[]=\_T("客户端......"); //要写入的数据

DWORD dwWrite;

if(!WriteFile(hPipe,buf,lstrlen(buf)+1,&dwWrite,NULL))

{return ;}

### 匿名管道

只能在父子进程间通信

#### 父进程

//用于读写的句柄

HANDLE hRead=NULL;

HANDLE hWrite=NULL;

析构时：

if(hRead){CloseHandle(hRead);}

if(hWrite){CloseHandle(hWrite);}

初始化:

SECURITY\_ATTRIBUTES sa;

sa.bInheritHandle=TRUE;

sa.lpSecurityDescriptor=NULL;

sa.nLength=sizeof(SECURITY\_ATTRIBUTES);

if(!CreatePipe(&hRead,&hWrite,&sa,0))//创建匿名管道

{ return ;}

//准备相关结构体

STARTUPINFO sui;

PROCESS\_INFORMATION pi;

ZeroMemory(&sui,sizeof(STARTUPINFO));

sui.cb=sizeof(STARTUPINFO);

sui.dwFlags=STARTF\_USESTDHANDLES;

sui.hStdInput=hRead;

sui.hStdOutput=hWrite;

sui.hStdError=GetStdHandle(STD\_ERROR\_HANDLE);

if(!CreateProcess("d:\\child.exe",NULL,NULL,NULL,TRUE,0,NULL,NULL,&sui,&pi))

{//创建子进程 假设子进程放在d盘，名为child.exe

CloseHandle(hRead);

CloseHandle(hWrite);

hRead=NULL;

hWrite=NULL;

return ;

}

else

{

CloseHandle(pi.hProcess);

CloseHandle(pi.hThread);

}

读取：

TCHAR buf[100];

DWORD dwRead;

if(!ReadFile(hRead,buf,100,&dwRead,NULL))

{return ;}

CString str;

str.Format(\_T("%s"),buf);

写入：

TCHAR buf[]=\_T("父进程......"); //要写入的数据

DWORD dwWrite;

if(!WriteFile(hWrite,buf,lstrlen(buf)+1,&dwWrite,NULL))

{return ;}

#### 子进程

//用于读写的句柄

HANDLE hRead=NULL;

HANDLE hWrite=NULL;

析构时：

if(hRead){CloseHandle(hRead);}

if(hWrite){CloseHandle(hWrite);}

初始化:

hRead=GetStdHandle(STD\_INPUT\_HANDLE);

hWrite=GetStdHandle(STD\_OUTPUT\_HANDLE);

读取：

TCHAR buf[100];

DWORD dwRead;

if(!ReadFile(hRead,buf,100,&dwRead,NULL))

{return ;}

CString str;

str.Format(\_T("%s"),buf);

写入：

TCHAR buf[]=\_T("子进程.....");

DWORD dwWrite;

if(!WriteFile(hWrite,buf,lstrlen(buf)+1,&dwWrite,NULL))

{return ;}

### 邮槽

无连接，单向通信，服务器读取，客户端写入。

#### 服务端

HANDLE hMailslot;

hMailslot=CreateMailslot(\_T("\\\\.\\mailslot\\MySlot"),0,MAILSLOT\_WAIT\_FOREVER,NULL);//第一个参数格式固定，只有最后的名字MySlot可以改变

if(INVALID\_HANDLE\_VALUE==hMailslot)

{return ;}

TCHAR buf[100];

DWORD dwRead;

if(!ReadFile(hMailslot,buf,100,&dwRead,NULL))

{

CloseHandle(hMailslot);

return ;

}

CString str;

str.Format(\_T("%s"),buf);

CloseHandle(hMailslot);

#### 客户端

HANDLE hMailslot;

hMailslot=CreateFile(\_T("\\\\.\\mailslot\\MySlot"),GENERIC\_WRITE,FILE\_SHARE\_READ,NULL,OPEN\_EXISTING,FILE\_ATTRIBUTE\_NORMAL,NULL);

if(INVALID\_HANDLE\_VALUE==hMailslot)

{return ;}

TCHAR buf[]=\_T("客户端.....");//要发送的数据

DWORD dwWrite;

if(!WriteFile(hMailslot,buf,lstrlen(buf)+1,&dwWrite,NULL))

{

CloseHandle(hMailslot);

return ;

}

CloseHandle(hMailslot);