# WINDOWS编程基础

## 应用程序分类

●控制台程序 ConsoleDOS程序,本身没有窗口,通过 Windows dos窗口执行

●窗口程序拥有自己的窗口,可以与用户交互

●库程序

存放代码、数据的程序,执行文件可以从中取岀代码执行和获取数据

静态库程序:扩展名LIB,在编译链接程序时,将代码放入到执行文件中。

动态库程序:扩展名DLL,在执行文件执行时从中获取代码。

应用程序对比

入口函数

控制台程序-main

窗口程序- Win Main

动态库程序-DLLMain

静态库程序-无入口函数

文件存在方式

控制台程序、窗口程序-EXE文件

动态库程序-DLL文件

静态库程序-LB文件

## 开发工具和库

编译工具

●编译器CL.EXE将源代码编译成目标代码.obj

●链接器LINK.EXE将目标代码、库链接生成最终文件

●资源编译器RC.EXE(r)将资源编译,最终通过链接器存入最终文件

路径: C: Program Files(x86)\Microsoft Visual Studio 10.0\VC\bin

库和头文件

● Windows库

kerne32dl-提供了核心的API,例如进程、线程、内存管理等。

user32.dl-提供了窗口、消息等

APIgdi32.d-绘图相关的API

路径:C: Windows\ System32

●头文件

windows. h-所有 windows头文件的集合

Widef.h- windows数据类型

Winbase.h-kernel32的API

wingdi.h-gdi32的API

Winuser.h-user32的API

winnt.h- UNICODE字符集支持

路径: C: Program Files(x86)\Microsoft SDKs\Windows\v70A\Include

相关函数

● int Winapi Winmair(

HINSTANCE hInstance,//当前程序的实例句柄

HINSTANCE hPrevInstance,/当前程序前一个实例句柄

LPSTR lpCmdLine//命令行参数字符串

int ncmdshow//窗口的显示方式

);

● int Messagebox(

HWND hWnd父窗口句柄

LPCTSTR IpText//显示在消息框中的文字

LPCTSTR IpCaption,//显示在标题栏中的文字

UINT uType//消息框中的按钮、图标显示类型

);//返回点击的按钮ID

程序编译过程

●编译环境准备

VCVARS32 BAT

●编译程序-CL

CL.EXE -C XXX C

●链接程序-LINK

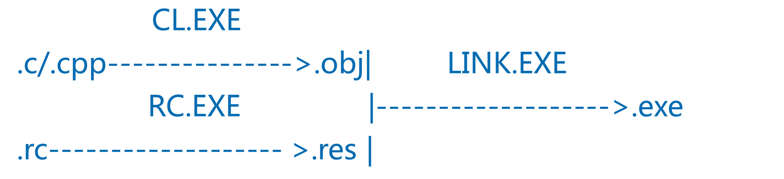
LINK EXE XXX. obj xXx.lib

●执行

●编写资源的文件-.rc资源脚本文件

●编译rc文件- RC.EXE

●将资源链接到程序中-LINK.EXE



## 创建窗口全过程

●定义 Win Main函数

●定义窗口处理函数(自定义处理消息)

●注册窗口类(向操作系统写入一些数据)

●创建窗口(内存中创建窗口)

●显示窗口(绘制窗口的图像)

●消息循环(获取/翻译/派发消息)

●消息处理

### 创建窗口

#include <windows.h>

//

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)

{

switch (message) {

case WM\_DESTROY:

PostQuitMessage(0);

break;

}

return DefWindowProc(hWnd, message, wParam, lParam);

}

int CALLBACK WinMain(HINSTANCE hInstance,

HINSTANCE hPrevInstance,

LPSTR lpCmdLine,

int nCmdShow)

{

WNDCLASS wc = { 0 };

wc.cbClsExtra = 0;

wc.cbWndExtra = 0;

wc.hbrBackground = (HBRUSH)(COLOR\_WINDOW + 1);

wc.hCursor = NULL;

wc.hIcon = NULL;

wc.hInstance = hInstance;

wc.lpfnWndProc = WndProc;

wc.lpszClassName = "Main";

wc.lpszMenuName = nullptr;

wc.style = CS\_HREDRAW | CS\_VREDRAW;

RegisterClass(&wc);

HWND hWnd = CreateWindowW(L"Main", L"window", WS\_OVERLAPPEDWINDOW,

100, 100, 500, 500, nullptr, nullptr, hInstance, nullptr);

ShowWindow(hWnd, SW\_SHOW);

UpdateWindow(hWnd);

MSG nMsg = { 0 };

while (GetMessage(&nMsg, nullptr, 0, 0))

{

TranslateMessage(&nMsg);

DispatchMessage(&nMsg); //派发消息

}

return 0;

}

### 细节

ATOM RegisterClass(

CONST WNDCLASS \*IpWndClass//窗囗类的数据

）；注册成功后，返回一个数字标识。

typedef struct \_WNDCLASS{

UINT style;//窗囗类的风格

WNDPROC IpfnWndProc；//窗口处理函数

int cbCIsExtra；//窗囗类的附加数据buff的大小

int cbWndExtra；//窗囗的附加数据buff的大小

HINSTANCE hInstance;/当前模块的实例句柄

HICON hIcon;//窗口图标句柄

HCURSOR hCursor;//鼠标的句柄

HBRUSH hbrBackground;//绘制窗口背景的画刷句柄

LPCTSTR lpszMenuName;/窗口菜单的资源ID字符串

LPCTSTR IpszClassName;/窗囗类的名称

}WNDCLASS,\*PWNDCLASS;

CreateWindow/CreateWindowEx)

HWND CreateWindowEx(

DWORD dwExStyle,//窗囗的扩展风格 CreateWindowEx比CreateWindow多此参数

LPCTSTR IpClassName,//已经注册的窗囗类名称

LPCTSTR IpWindowName,//窗口标题栏的名字

DWORD dwStyle,//窗囗的基本风格

int x,/窗囗左上角水平坐标位置

int y,//窗口左上角垂直坐标位置

int nWidth,//窗囗的宽度

int nHeight,/窗囗的高度

HWND hWndParent,//窗口的父窗口句柄

HMENU hMenu,//窗口菜单句柄

HINSTANCE hInstance,//应用程序实例句柄

LPVOID IpParam//窗囗创建时附加参数

）；创建成功返回窗口句

### 创建子窗口时

要设置父窗口句柄·

创建风格要增加WS\_CHILD | WS\_VISIBLE

消息队列的概念

●消息队列是用于存放消息的队列。

●消息在队列中先入先出。

●所有窗口程序都具有消息队列。

●程序可以从队列中获取消息。

## 字符编码

编码历史背景

●ASC ●ASCI ●DBCS ● UNICODE

### DBCS和 UNICODE码

●DBCS字符编码

A 我 是 程 序 员

01 0203 0405 0607 0809 0A0B

但是解析时,可能为:

01 0203 0405 0607 0809 0A0B

0102 0304 0506 0708 090A 0B

●UNICODE编码

A 我 是 程 序 员

0001 0203 0405 0607 0809 0A0B

●不存在解析的问题

### 宽字节字符

● wchar\_t每个字符占2个字节

char每个字符占1个字节

wchar\_t实际是 unsigned short类型,定义时,需要增加“L",通知编译器按照双字节编译字符串,采用∪ NICODE编码。

●需要使用支持 wchar\_t函数操作宽字节字符串。例如

wchar\_t\* pwszText =L”Hello wchar”;

wprintf(L" %s\n", pwszText );

### TCHAR

#ifdef UNiCode

typedef wchar\_t TCHAR,;

#define\_ TEXT(quote)L##quote

#else

typedef char TCHAR

#define\_TEXT(quote)quote

#endif

### 打印 UNICODE字符

UNICODE字符打印

wprintf对 UNICODE字符打印支持不完善

在 Windows下使用 Write Console apl打印 UNICODE字符 GetStdHandle

# WINDOWS消息

## 消息组成（windows平台下）

窗口句柄

消息ID

消息的两个参数（两个附带信息）

消息产生的时间

消息产生时的鼠标位置

## 消息队列

### 消息队列的概念

●消息队列是用于存放消息的队列。

●消息在队列中先入先出。

●所有窗口程序都具有消息队列

●程序可以从队列中获取消息

### 消息队列的分类

●系统消息队列-由系统维护的消息队列。存放系统产生的消息,例如鼠标、键盘等。

●程序消息队列-属于每一个应用程序(线程)的消息队列。由应用程序(线程)维护

### 消息和消息队列的关系

消息和消息队列的关系

1当鼠标、键盘产生消息时,会将消息存放到系统消息队列

2系统会根据存放的消息,找到对应程序的消息队列。

3将消息投递到程序的消息队列中。

根据消息和消息队列之间使用关系,将消息分成两类:

队列消息-消息的发送和获取,都是通过消息队列完成。

非队列消息-消息的发送和获取,是直接调用消息的窗口处理完成。

●队列消息-消息发送后,首先放入队列,然后通过消息循环,从队列当中获取。

GetMessage-从消息队列中获取消息

PostMessage-将消息投递到消息队列

常见队列消息: WM PAINT、键盘、鼠标、定时器。

●非队列消息-消息发送时,首先查找消息接收窗口的窗口处理函数,直接调用处理函数,完成消息。

SendMessage-直接将消息发送给窗口的处理函数,并等候处理结果

常见消息: WM CREATE、 WM SIZE等

### 深谈 GetMessage

●在程序(线程)消息队列查找消息,如果队列有消息,检査消息是否满足指定条件( HWNDID范围),不满足条件就不会取出消息,否则从队列取出消息返回。

●如果程序(线程)消息队列没有消息,向系统消息队列获取属于本程序的消息。如果系统队列的当前消息属于本程序,系统会将消息转发到程序消息队列中。

●如果系统消息队列也没有消息,检査当前进程的所有窗口的需要重新绘制的区域,如果发现有需要绘制的区域,产生 WM PAINT消息,取得消息返回处理。

●如果没有重新绘制区域,检查定时器如果有到时的定时器,产生 WM TIMER返回处理执行如果没有到时的定时器,整理程序的资源、内存等等。

●GetMessage会继续等候下一条消息。 PeekMessage会返回 FALSE,交出程序的控制权

●注意: GetMessage如果获取到是 WM QUIT函数会返回 FALSE

### WM PAINT消息

●产生时间:当窗口需要绘制的时候。

●附带信息:

WPara:0。

IParam: 0。

●专职用法:用于绘图。

●窗口无效区域:需要重新绘制的区域。

BOOL Invalidate Rect(

HWND hWnd,/口句柄

CONST RECT\* IrEct,/区域的矩形坐标

bool eRase/重绘前是否先擦除

）；

●消息处理步骤

1. 开始绘图

HDC BeginPaint(

HWND hwnd, //绘图窗口

LPPAINTSTRUCT IpPaint //绘图参数的BUFF

);返回绘图设备句柄HDC

1. 正式绘图
2. 结束绘图

BOOL EndPaint(

HWND hWnd, //绘图窗口

CONST PAINTSTRUCT \*IpPaint //绘图参数的指针

BeginPaint返回

);

## 键盘消息

### 键盘消息分类

WM\_KEYDOWN-按键被按下时产生

WM\_KEYUP-按键被放开时产生

WM-SYSKEYDOWN-系统键按下时产生比如ALT, F10

WM-SYSKEYUP-系统键放开时产生

附带信息

WPARAM-按键的Virtual Key

LPARAM-按键的参数,例如按下次数

### 字符消息(WM\_CHAR )

TranslateMessage在转换WM\_KEYDOWN消息时,对于可见字符可以产生WM\_CHAR,不可见字符无此消息,

附带信息:

WPARAM-输入的字符的ASCI字符编码值

LPARAM-按键的相关参数

### 鼠标消息

鼠标消息分类

基本鼠标消息

WM\_LBUTTONDOWN-鼠标左键按下

WM\_LBUTTONUP-鼠标左键抬起

WM\_RBUTTONDOWN-鼠标右键按下

WM\_RBUTTONUP-鼠标右键抬起

WM\_MOUSEMOVE-鼠标移动消息

双击消息

WM\_LBUTTONDBLCLK-鼠标左键双击

WM\_RBUTTONDBLCLK-鼠标右键双击

滚轮消息

WM\_MOUSEWHEEL-鼠标滚轮消息

鼠标基本消息

附带信息:

WPARAM :其他按键的状态,例如Ctrl/Shift等

IPARAM :鼠标的位置,窗口客户区坐标系。

LOWORD X坐标位置

HIWORD Y坐标位置

一般情况鼠标按下/抬起成对出现。在鼠标移动过程中,会根据移动速度产生一系列的WM\_MOUSEMOVE消息。

### 鼠标双击消息

附带信息

wPARAM-其他按键的状态,例如Ctrl/Shift等

IPARAM-鼠标的位置,窗口客户区坐标系。

LOWORD(Param) //X坐标位置

HIWORD(IParam) //坐标位置

消息产生顺序

以左键双击为例:

WM\_LBUTTONDOWNW

WM\_LBUTTONUP

WM\_LBUTTONDBLCLK

WM\_LBUTTONUP

使用时需要在注册窗口类的时候添加CS-DBLCLKS风格。

### 鼠标滚轮

鼠标滚轮消息

附带信息:

WPARAM

LOWORD-其他按键的状态

HIWORD-滚轮的偏移量,通过正负值表示表示滚动方向。

正:向前滚动 负:向后滚动

IPARAM :鼠标当前的位置,屏幕坐标系

LOWORD-X坐标

HIWORD-Y坐标

使用：

通过偏移量,获取滚动的方向和距离。

### 定时器消息

定时器消息介绍

产生时间:

在程序中创建定时器,当到达时间间隔时,定时器会向程序发送一个WM\_TIMER消息。定时器的精度是毫秒,但是准确度很低。例如设置时间间隔为1000ms ,但是会在非1000毫秒到达消息。

附带信息:

WPARAM :定时器ID

IPARAM :定时器处理函数的指针

创建和销毁定时器

创建定时器

UINT PTR SetTimer(

HWND hWnd,//定时器窗口句柄

UINT\_PTR nIDEvent, //定时器

IDUINT uElapse,//时间间隔

TIMERPROC IpTimerFunc //定时器处理函数指针(一般不使用,为NULL

));创建成功,返回非0。

关闭定时器

BOOL KillTimer(

HWND hWnd,//定时器窗口句柄

UINT\_PTR uIDEvent //定时器ID

);

## 菜单资源

菜单分类

窗口的顶层菜单

弹出式菜单

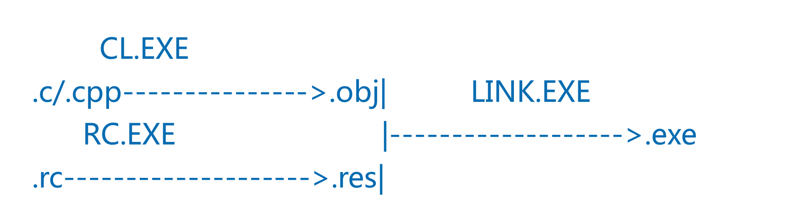
系统菜单

HMENU类型表示菜单, ID表示菜单项。

资源相关

资源脚本文件： \*rc文件

编译器: RC.EXE



菜单资源的使用

添加菜单资源

加载菜单资源

1. 注册窗口类时设置菜单
2. 创建窗口传参设置菜单
3. 在主窗口WM\_CREATE消息中利用SetMenu函数设置菜单

加载菜单资源

HMENU LoadMenu(

HINSTANCE hInstance, // handle to module

LPCTSTR IpMenuName // menu name or resource identifier

);

### 命令消息(WM-COMMAND )处理

附带信息:

WPARAM

HIWORD -ХT菜单为0

LOWORD -单项的ID

IPARAM-对于菜单为0

### 上下文菜单

显示上下文菜单

BOOL TrackPopupMenu(

HMENU hMenu, //菜单句柄

UINT uFlags, //显示方式

int x.//平位置,屏幕坐标系

int y.//垂直位置,屏幕坐标系

int nReserved, /保留,必须0

HWND hWnd, //处理菜单消息的窗口句柄

CONST RECT \*prcRect //NULL,忽略

); TrackPopupMenu是阻塞函数

WM\_RBUTTONUP

鼠标右键弹起消息为窗口坐标系坐标,要使用需将其转换成屏幕坐标系坐标ClientToScreen

WM CONTEXTMENU

wParam :右键点击的窗口句柄

IPARAM : LOWORD X坐标,屏幕坐标系

HIWORD Y坐标,屏幕坐标系

WM\_CONTEXTMENU消息是在WM\_RBUTTONUP消息之后产生。

# Windows资源

## 图标资源

添加资源

注意图标的大小,一个图标文件中,可以有多个不同大小的图标。

加载

HICON LoadIcon(

HINSTANCE hInstance, // handle to application instance

LPCTSTR IpIconName // name string or resource identifier

);成功返回HICON句柄

设置

注册窗口类

## 字符串资源

添加字符串资源

添加字符串表,在表中增加字符串

字符串资源的使用

int LoadString(

HINSTANCE hInstance, // handle to resource moduleUINT uID, //字符串

IDLPTSTR IpBuffer, //存放字符串BUFF

int nBufferMax //字符串BUFF长度

);成功返回字符串长度,失败0

## 光标资源

添加光标的资源

光标的大小默认是32X32像素,每个光标有HotSpot ,是当前鼠标的热点

加载资源

HCURSOR LoadCursor(

HINSTANCE hInstance, // handle to application instance

LPCTSTR IpCursorName // name or resource identifier

); hInstance-可以为NULL,获取系统默认的Cursor

设置资源

在注册窗口时,设置光标

使用SetCursor设置光标

HCURSOR SetCursor(

HCURSOR hCursor / handle to cursor );

WM\_SETCURSOR消息参数 WPARAM-当前使用的光标句柄

IPARAM-LOWORD当前区域的代码( Hit-Test code

HTCLIENT/ HTCAPTION...

HIWORD-当前鼠标消息ID

## 加速键资源

添加 资源添加加速键表,增加命令ID对应的加速键。

使用

加载加速键表

HACCEL LoadAccelerators(

HINSTANCE hInstance, // handle to module

LPCTSTR IpTableName // accelerator table name

);返回加速键表句柄

翻译加速键

int TranslateAccelerator(

HWND hWnd,//处理消息的窗口句柄

HACCEL hAccTable, //加速键句柄 LPMSG IpMsg //消息

);如果是加速键,返回非零。

在WM\_COMMAND中相应消息,消息参数

WPARAM : HIWORD为1表示加速键,为0表示菜单。

LOWORD为命令ID。

IParam :为0

## 绘图编程

绘图基础

绘图设备DC ( Device Context ) ,绘图上下文/绘图描述表’

HDC-DC句柄,表示绘图设备

GDI -Windows graphics device interface (Win32提供的绘图API )

颜色

计算机使用红、绿、蓝

R-0~255

G-0~255

B-0~255

每一个点颜色是3个字节24位保存0-2^24-1

16位:5,5,6

32位:8,8,8, 8绘图或透明度

颜色的使用

COLORREF-实际DWORD 例如: COLORREF nColor =0;

赋值使用RGB宏 例如: nColor =RGB( 0, 0, 255);

获取RGB值

GetRValue/GetGValue/GetBValue

例如: BYTE nRed = GetRValue( nColor );

基本图形绘制

SetPixel设置指定点的颜色’

COLORREF SetPixel(

HDC hdc,//DC句柄

int X,//X坐标

int Y//Y坐标

COLORREF crColor //设置的颜色

);返回点原来的颜色

线的使用(直线、弧线)

MoveToEx-指名窗口当前点

LineTo-从窗口当前点到指定点绘制一条直线

当前点:上一次绘图时的最后一点,初始为(0,0)点。

封闭图形:能够用画刷填充的图形

Rectangle / Ellipse

## GDI绘图对象

### GDI绘图对象-画笔

画笔的作用

线的颜色、线型、线粗。

HPEN -画笔句柄

画笔的使用

1创建画笔

HPEN CreatePen(

int fnPenStyle, //画笔的样式

int nWidth, //画笔的粗细

COLORREF crColor //画笔的颜色

);创建成功返回句柄

PS-SOILD-实心线,可以支持多个像素宽其他线型只能是一个像素宽。

2将画笔应用到DC中

HGDIOBJ SelectObject(

HDC hdc,//绘图设备句柄

HGDIOBJ hgdiobj//

GDI绘图对象句柄,画笔句柄

);返回原来的GDI绘图对象句柄注意保存原来DC当中画笔。

3绘图

4取出DC中的画笔

将原来的画笔,使用SelectObject函数,放入到设备DC中,就会将我们创建的画笔取出。

5释放画笔

BOOL DeleteObject(

HGDIOBJ hObject //GDI绘图对象句柄,画笔句柄

);只能删除不被DC使用的画笔,所以在释放前,必须将画笔从DC中取出。

### GDI绘图对象-画刷

画刷相关

画刷-封闭图形的填充的颜色、图案

HBRUSH-画刷句柄

画刷的使用

1创建画刷

CreateSolidBrush-创建实心画刷

CreateHatchBrush-创建纹理画刷

2将画刷应用到DC中SelectObject

3绘图

4将画刷从DC中取出

SelectObject

5删除画刷

DeleteObject

其他

可以使用GetStockObject函数获取系统维护的画刷、画笔等。

如果不使用画刷填充,需要使用NULL-BRUSH参数,获取不填充的画刷。

GetStockObject返回的画刷不需要DeleteObject.

## 位图

位图绘制

位图相关

光栅图形-记录图像中每一点的颜色等信息。

矢量图形-记录图像算法、绘图指令等。

HBITMAP -位图句柄

位图的使用

1在资源中添加位图资源

2从资源中加载位图LoadBitmap

3创建一个与当前DC相匹配的DC (内存DC)

HDC CreateCompatibleDC(

HDC hdc //当前DC句柄,可以为NULL (使用屏幕DC)

);返回创建好的DC句柄

4将位图放入匹配的DC中SelectObject

5成像(1:1) BOOL BitBlt(

HDC hdcDest, //目的DC

int nXDest, //目的左上X坐标

int nYDest, //目的左上Y坐标

int nWidth, //目的宽度

int nHeight, //目的高度

HDC hdcSrc, //源DC

int nXSrc, //源左上X坐标

int nYSrc, //源左上Y坐标

DWORD dwRop //成像方法SRCCOPY

)；

缩放成像

B00L StretchBlt(

HDC hdcDest, / handle to destination DC

int nXOriginDest, // x-coord of destination upper-left corner

int nYOriginDest, //y-coord of destination upper-left corner

int nWidthDest, // width of destination rectangle

int nHeightDest, // height of destination rectangle

HDC hdcSrc, / handle to source DC

int nXOriginSrc, //x-coord of source upper-left corner

int nYOriginSrc, // y-coord of source upper-left corner

int nWidthSrc, //源DC宽

int nHeightSrc, //源DC高

DWORD dwRop // raster operation code

）；

6取出位图

SelectObject

7释放位图

DeleteObject

8释放匹配的DC

DeleteDC

## 文本绘制

### 绘制字符串

文字的绘制

TextOut-将文字绘制在指定坐标位置。

int DrawText(

HDC hDC, //DC句柄

LPCTSTR IpString, //字符串

int nCount, //字符数量

LPRECT IpRect, //绘制文字的矩形框

UINT uFormat //绘制的方式

);

文字颜色和背景

文字颜色: SetTextColor

文字背景色: SetBkColor

文字背景模式: SetBkMode (OPAQUE / TRANSPARENT)

### 字体

字体相关

Window常用的字体为TrueType格式的字体文件

字体名-标识字体类型

HFONT -字体句柄

字体的使用

1创建字体

HFONT CreateFont(

int nHeight,//字体高度

int nWidth,//字体宽度

int nEscapement,//字符串倾斜角度

int nOrientation,//字符旋转角度 （Z轴）

int fnWeight,//字体的粗细

DWORD fdwltalic,//斜体

DWORD fdwUnderline,//字符下划线

DWORD fdwStrikeOut,//删除线

DWORD fdwCharSet,//字符集

DWORD fdwOutputPrecision,//输出精度

DWORD fdwClipPrecision,//剪切精度

DWORD fdwQuality,/输出质量

DWORD fdwPitchAndFamily,/匹配字体

LPCTSTR IpszFace//字体名称 ）；

2应用字体到

DC SelectObject

3绘制文字

DrawText/TextOut

4取出字体

SelectObject

5删除字体

DeleteObject

# Windows内核开发

## 对话框

### 对话框原理

对话框的分类

模式对话框-当对话框显示时,会禁止其他窗口和用户交互操作。

无模式对话框-在对话框显示后,其他窗口仍然可以和用户交互操作。

对话框基本使用

1对话框窗口处理函数

2注册窗口类(不使用)

3创建对话框

4对话框的关闭

对话框窗口处理函数(并非真正的对话框窗口处理函数)

INT CALLBACK DialogProc(

HWND hwndDlg, //窗口句柄

UINT uMsg, //消息

IDWPARAM wParam, //消息参数

LPARAM IParam //消息参数

)；

返回TRUE-缺省处理函数不需要处理。

返回FALSE-交给缺省处理函数处理。

不需要调用缺省对话框窗口处理函数。

### 模式对话框

创建对话框

INT DialogBox(

HINSTANCE hInstance,//应用程序实例句柄

LPCTSTR IpTemplate, //对话框资源

IDHWND hWndParent, //对话框父窗口

DLGPROC IpDialogFunc //自定义函数

);

DialogBox是一个阻塞函数,只有当对话框关闭后,才会返回,继续执行后续代码。

返回值是通过EndDialog设置

对话框的关闭

BOOL EndDialog(

HWND hDlg,//关闭的对话框窗口句柄

INT\_PTR nResult //关闭的返回值

);

关闭模式对话框,只能使用EndDialog ,不能使用DestroyWindow等函数。

nResult是DialogBox函数退出时的返回值。

对话框的消息

WM INITDIALOG-对话框创建之后显示之前,通知对话框窗口处理函数,可以完成自己的初始化相关的操作。

### 非模式对话框：

创建对话框

HWND CreateDialog(

HINSTANCE hInstance,//应用程序实例句柄

LPCTSTR IpTemplate,//模板资源

IDHWND hWndParent,//父窗口

DLGPROC IpDialogFunc/自定义函数)

非阻塞函数，创建成功返回窗口句柄，需要使用ShowWindow函数显示,

对话框的关闭

关闭时使用DestroyWindow销毁窗口，不能使用EndDialog关闭对话框。

对话框的消息

WM-INITDIALOG-对话框创建之后显示之前,通知对话框窗口处理函数可以完成自己的初始化相关的操作。

## 静动态库

### 静态库

静态库特点

运行不存在。

静态库源码被链接到调用程序中。

目标程序的归档。

C语言静态库

C静态库的创建

1)创建一个静态库项目。

2)添加库程序,源文件使用C文件。

C静态库的使用

库路径设置:可以使用pragma关键字设置

#pragma comment(lib, "../lib/clib.lib"

C++语言静态库

C++静态库的创建

1)创建一个静态E项目。

2)添加库程序,源文件使用CPP文件。

C++静态库的使用

库路径设置:可以使用pragma关键字设置

#pragma comment( lib, ../ib/cpplib.lib")

### 动态库

动态库特点·动态库特点

1）运行时独立存在

2）源码不会链接到执行程序

3）使用时加载（使用动态库必须使动态库执行）

与静态库的比较：

1）由于静态库是将代码嵌入到使用程序中，多个程序使用时，会有多份代码，所以代码体积会增大。动态库的代码只需要存在一份，其他程序通过函数地址使用，所以代码体积小。

2）静态库发生变化后，新的代码需要重新链接嵌入到执行程序中。动态库发生变化后，如果库中函数的定义（或地址）未变化，其他使用DLL的程序不需重新链接。

动态库的创建·

创建动态库项目

添加库程序

库程序导出-提供给使用者库中的函数等信息。

1）声明导出：使用\_declspec(dllexport)导出函数注意：动态库编译链接后，也会有LIB文件，是作为动态库函数映射使用，与静态库不完全相同。(此方法编译修改函数名)

2）模块定义文件.def例如：(此方法编译不修改函数名)

LIBRARY DLLFunc//库

EXPORTS //库导出表

DLL\_Mul @1//导出的函数

动态库的使用：

隐式链接（操作系统负责使动态库执行）

1）头文件和函数原型可以在函数原型的声明前，增加\_declspec(dllimport)

2）导入动态库的Lib文件

3）在程序中使用函数

4）隐式链接的情况，dll文件可以存放的路径：

（1）与执行文件中同一个目录下

（2)当前工作目录

（3）Windows目录

（4）Windows/System32目录

（5)Windows/System

（6）环境变量PATH指定目录

显式链接（程序员自己负责使动态库执行）

1）定义函数指针类型 typedef

2）加载动态库HMODULE LoadLibrary(LPCTSTR IpFileName//动态库文件名或全路径）；返回DLL的实例句柄（HINSTANCE)

3）获取函数地址FARPROC GetProcAddress(HMODULE hModule,//DLL句柄LPCSTR IpProcName //函数名称）；成功返回函数地址

4）使用函数

5）卸载动态库BOOL FreeLibrary(HMODULE hModule //DLL的实例句柄

动态库中封装类 ·

在类名称前增加\_declspec(dllexport)定义，例如：

class\_declspec(dllexport)CMath{

...

}

通常使用预编译开关切换类的导入导出定义，例如：

#ifdef DLLCLASS\_EXPORTS

#define EXT\_CLASS \_declspec(dllexport)//DLL ori

#else

#define EXT\_CLASS \_declspec(dllimport)/使用者

#endif

class EXT\_CLASS CMath{

...

}

# Windows线程开发

## 线程基础

Windows线程是可以执行的代码的实例。系统是以线程为单位调度程序。一个程序当中可以有多个线程，实现多任务的处理。

Windows线程的特点：

1）线程都具有1个ID

2）每个线程都具有自己的内存栈3)同一进程中的线程使用同一个地址空间。

线程的调度：

将CPU的执行时间划分成时间片，依次根据时间片执行不同的线程。

线程轮询：线程A->线程B->线程A..…

创建线程

创建线程 HANDLE CreateThread(

LPSECURITY\_ATTRIBUTES IpThreadAttributes,//安全属性 （废弃）

SIZE\_T dwStackSize,//线程栈的大小

LPTHREAD\_START\_ROUTINE IpStartAddress,//线程处理函数的函数地址

LPVOID IpParameter,/传递给线程处理函数的参数

DWORD dwCreationFlags,//线程的创建方式，0立即启动、CREATE\_SUSPENDED 手动

LPDWORD IpThreadld ∥创建成功，返回线程的ID ）；

创建成功，返回线程句柄

定义线程处理函数

DWORD WINAPI ThreadProc(

LPVOID IpParameter//创建线程时，传递给线程的参数

）；

销毁线程 ·

挂起

DWORD SuspendThread(

HANDLE hThread //handle to thread

）； ·

唤醒

DWORD ResumeThread(

HANDLE hThread //handle to thread

);

结束指定线程

BOOL Terminate Thread(

HANDLE hThread,// handle to thread

DWORD dwExitCode // exit code ·

)

结束函数所在的线程 (函数指ExitThread)

VOID ExitThread(

DWORD dwExitCode //exit code for this thread

）；

线程相关操作 ·

获取当前线程的ID

GetCurrentThreadld ·

获取当前线程的句柄

GetCurrentThread ·

等候单个句柄有信号 （只能用于有信号句柄）

VOID WaitForSingleObject(

HANDLE handle,//句柄BUFF的地址

DWORD dwMilliseconds //等候时间 (INFINITE为无线时间)

）；

同时等候多个句柄有信号

DWORD WaitForMultipleObjects(

DWORD nCount,/∥句柄数量

CONST HANDLE \*IpHandles,//句柄BUFF的地址

BOOL bWaitAll,//等候方式

DWORD dwMilliseconds //等候时间INFINITE

）；

bWaitAll-等候方式

TRUE-表示所有句柄都有信号，才结束等候

FASLE-表示句柄中只要有1个有信号，就结束等候。

## 原子锁·

相关问题

多个线程对同一个数据进行原子操作，会产生结果丢失。比如执行++运算时。

·错误代码分析：

当线程A执行g\_value++时，如果线程切换时间正好是在线程A将值保存到g\_value之前，线程B继续执行g value++，那么当线程A再次被切换回来之后，会将原来线程A保存的值保存到g\_value上，线程B进行的加法操作被覆盖。

·使用原子锁函数

InterlockedIncrement

InterlockedDecrement

InterlockedCompareExchange

InterlockedExchange

......

原子锁的实现：

直接对数据所在的内存操作，并且在任何一个瞬间只能有一个线程访问。

原子锁优缺点：

优：保证数据准确、所有锁里面数度最快的

缺：只能对运算符有用、一个函数只对应一个运算操作、

## 互斥·

相关的问题

多线程下代码或资源的共享使用。

互斥的使用

1创建互斥HANDLE CreateMutex(

LPSECURITY\_ATTRIBUTES IpMutexAttributes,//安全属性

BOOL bInitialOwner,//初始的拥有者

TRUE/FALSELPCTSTR IpName /命名

）；创建成功返回互斥句柄

2等候互斥 WaitFor..…互斥的等候遵循谁先等候谁先获取。

3释放互斥 BOOL ReleaseMutex(

HANDLE hMutex //handle to mutex

)

4关闭互斥句柄 CloseHandle

\*互斥里的函数同一时间只能被一个线程使用、互斥句柄为信号句柄（工作时无信号，Relese释放后有信号）

## 事件·

相关问题

程序之间的通知的问题。

事件的使用

1创建事件HANDLE CreateEvent(

LPSECURITY\_ATTRIBUTES IpEventAttributes,//安全属性

BOOL bManualReset,/事件重置（复位）方式，TRUE手动，FALSE自动

BOOL bInitialState,//事件初始状态，TRUE有信号

LPCTSTR IpName//事件命名

）；创建成功返回事件句柄

2等候事件

， WaitForSingleObject/WaitForMultipleObjects

3触发事件（将事件设置成有信号状态）

BOOL SetEvent( HANDLE hEvent //handle to event ）；

4复位事件（将事件设置成无信号状态）

BOOL ResetEvent( HANDLE hEvent // handle to event ）；

5关闭事件

CloseHandle

小心事件的死锁。

## 信号量

相关的问题

类似于事件，解决通知的相关问题。但提供一个计数器，可以设置次数。

信号量的使用

1创建信号量

HANDLE CreateSemaphore(

LPSECURITY\_ATTRIBUTES IpSemaphoreAttributes,//安全属性

LONG IInitialCount,//初始化信号量数量

LONG IMaximumCount,/信号量的最大值

LPCTSTR IpName /命名

）；创建成功返回信号量句柄

2等候信号量

WaitFor..每等候通过一次，信号量的信号减1，直到为0阻塞

3给信号量指定计数值

BOOL ReleaseSemaphore(

HANDLE hSemaphore,//信号量句柄

LONG IReleaseCount,//释放数量（即重新设置初始化信号量数量，不能超过最大值）

LPLONG IpPreviousCount/释放前原来信号量的数量，可以为NULL）；

4关闭句柄 CloseHandle