

# 第8章 Windows应用程序设计

- Ø Win32 API

- Ø Windows应用程序设计模式

- Ø Windows应用程序的基本结构

- Ø 结构化异常处理

- Ø 动态链接库

# Windows API与MFC

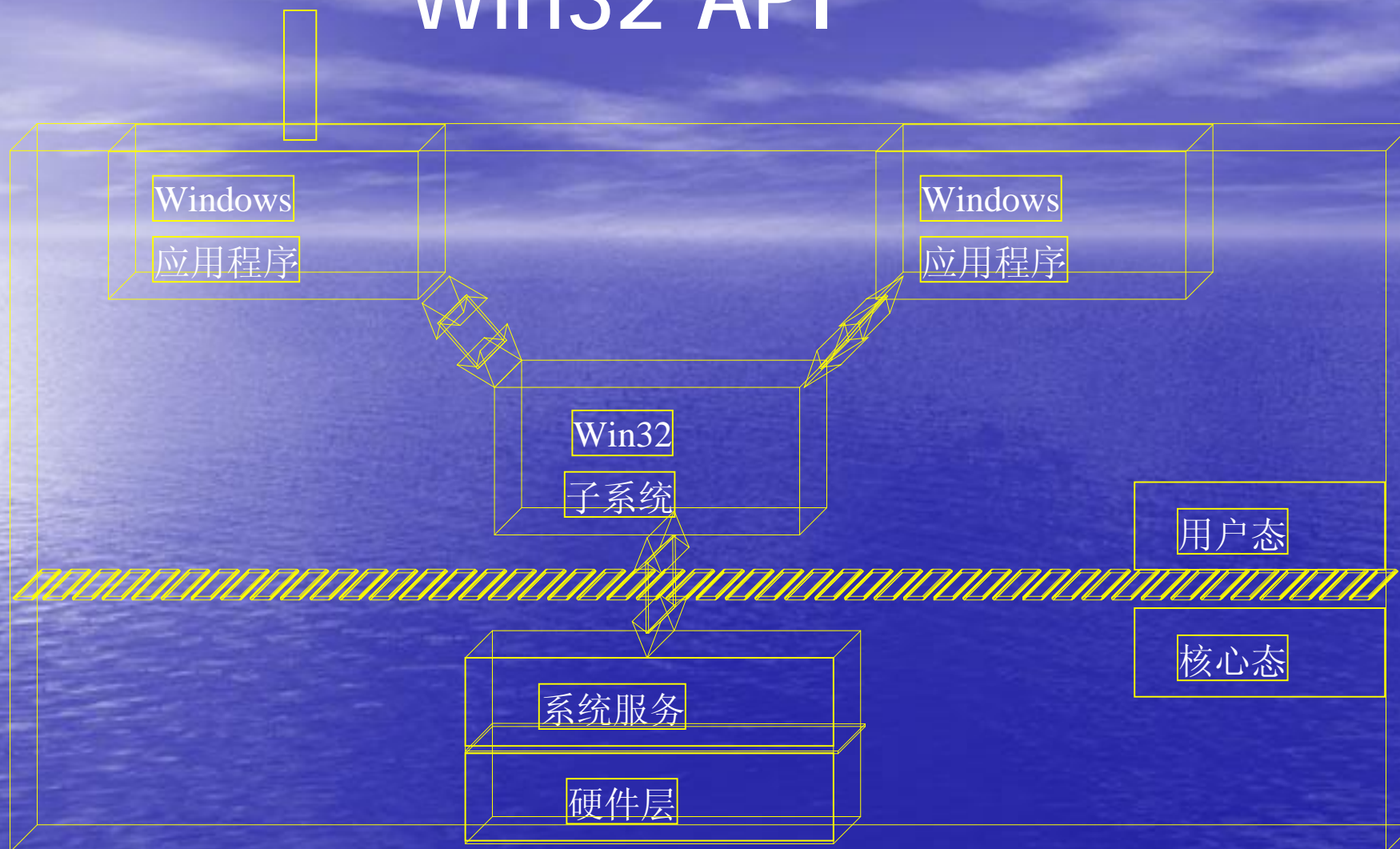
Ø早期是用C语言直接调用Windows SDK (Software Development Kit软件开发工具包)中的API (Application Program Interface 应用程序接口) 来开发Windows应用程序，由于需要亲自处理很多繁琐的编程细节，加上Windows API有两千多个函数、几百个数据结构和变量类型，所以进行Windows 编程是一件非常困难与痛苦的事。

ØVisual C++采用OOP来进行Windows 编程，将API的几千个函数、结构和变量类型封装在200个左右的类中（常用的只有十几个类），并且屏蔽掉了许多繁琐的编程细节，使得Windows 编程大大简化。微软公司称这些类所组成的类库为MFC (Microsoft Foundation Class Library微软基础类库)。

ØWindows操作系统依靠一组用户态环境子系统，作为应用程序与操作系统核心之间的接口



# Win32 API



Windows 应用程序与操作系统的关系

# Win32 API

USER32.DLL: 负责处理用户接口 标准Win32 API函数分类:

GDI32.DLL: 负责在图形设备上执行绘图操作

KERNEL32.DLL: 操作系统核心功能服务

- COMCTL32.DLL: 通用控件库
- COMDLG32.DLL: 公共对话框
- SHELL32.DLL: 用户界面外壳
- DIBENG.DLL: 图形引擎
- NETAPI32.DLL: 网络

- 系统服务
- 通用控件库
- GDI
- 网络服务
- 用户接口
- 系统Shell
- Windows 系统信息

Win32 API是用于Microsoft Windows操作系统家族的主要编程接口，在本章中指的是函数的核心集，其内容覆盖了进程、线程、内存管理、I/O、窗口和图形等范畴。Win32 API函数是文档化的、可调用的子程序，如CreateProcess、CreateFile、GetMessage等。



# Windows编程的特点

1. 面向对象的编程思想
2. 消息/事件驱动的程序结构
3. 资源共享与数据交换
4. 与设备无关GDI编程

# Windows编程工具

- ① Visual Basic: Microsoft公司于1991年推出, 可视化编程、简单实用、模块构建式、用户级, 功能有限、灵活性较差、Basic无标准/VB版本变化大/兼容性差, 适用于小型/简单的Windows软件的编程。(当前最新版.Net[7.0])
- ② Delphi: Borland公司于1995年推出, 内嵌语言为Pascal, 特点似VB, 数据库/服务器编程能力较强。(当前最新版7.0)
- ③ C++Builder: Borland公司于1991年推出Borland C++ (OWL), 1997年推出基于组件的RAD环境C++Builder, 使用类似VB与Delphi。(当前最新版6.0)
- ④ Visual C++: Microsoft公司于1993年推出的面向对象的Windows集成开发环境IDE (当前最新版.Net[7.0])



# Windows应用程序设计模式

Ø在字符界面型OS（如DOS）中执行应用程序时，程序必须取得CPU的控制权，整个运行过程都由程序本身来控制，称之为过程驱动的程序结构。

ØGUI型OS（如Windows2000/xp），情形则完全不同：在应用程序的运行过程中，大部分时间是由OS掌握控制权，只是在发生用户或系统事件（如移动鼠标、按下键盘、选择菜单或时钟、通信）后，OS才调用程序中的对应事件处理模块，所以称之为事件驱动的程序结构。

# Windows应用程序设计模式

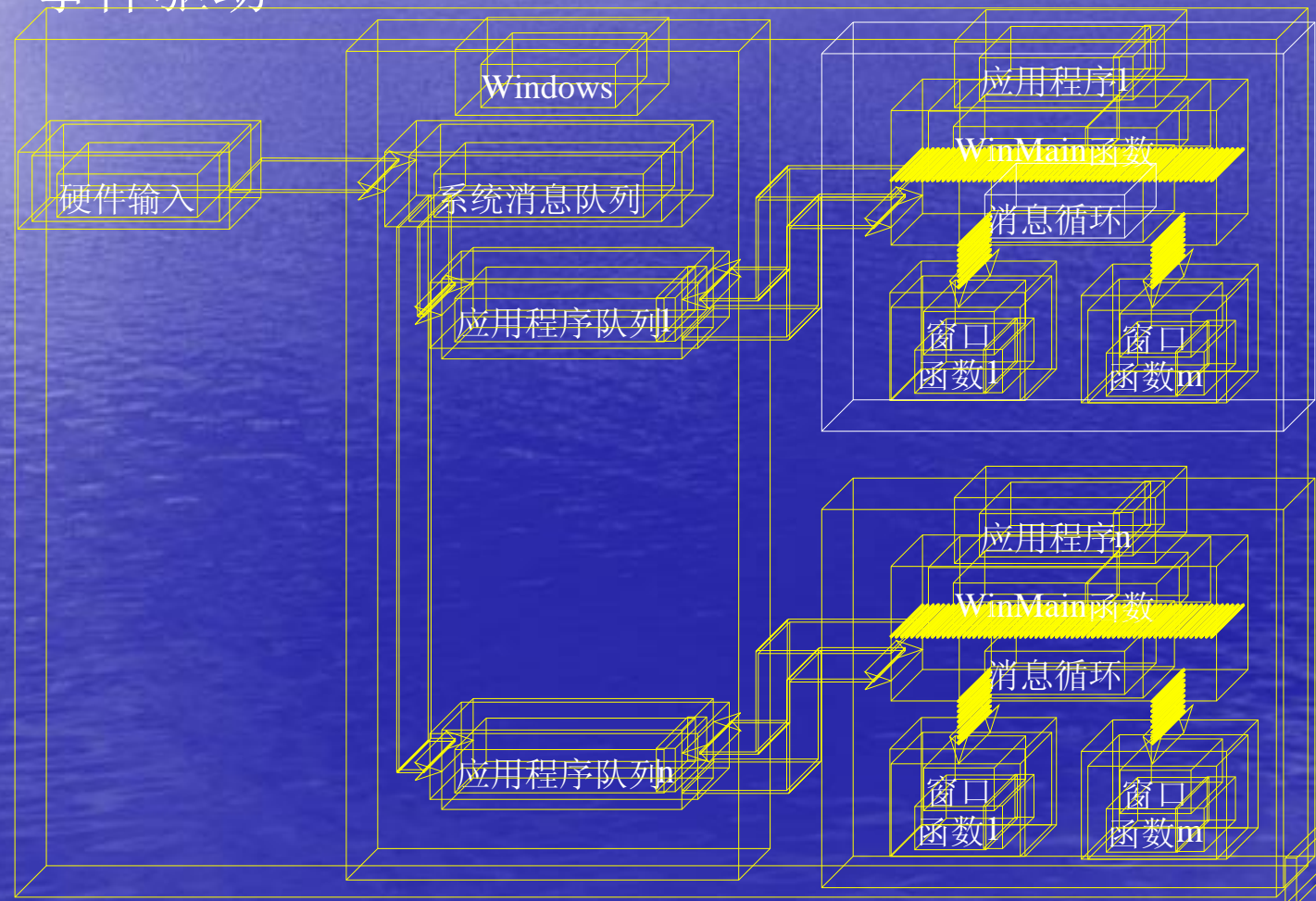
Ø事件驱动在GUI型OS中，事件发生后会产生对应的消息，消息提供了应用程序与OS之间、应用程序与应用程序之间的通信手段；多数应用程序的大部分代码都是用来响应和处理这些消息，所以也称之为消息驱动的程序结构。

ØWindows 维护着一个系统消息队列，Windows也为每个应用程序创建一个应用消息队列，事件所产生的消息，首先进入系统消息队列，然后再被传送到对应的应用消息队列，最后才被发送到消息所对应的窗口。

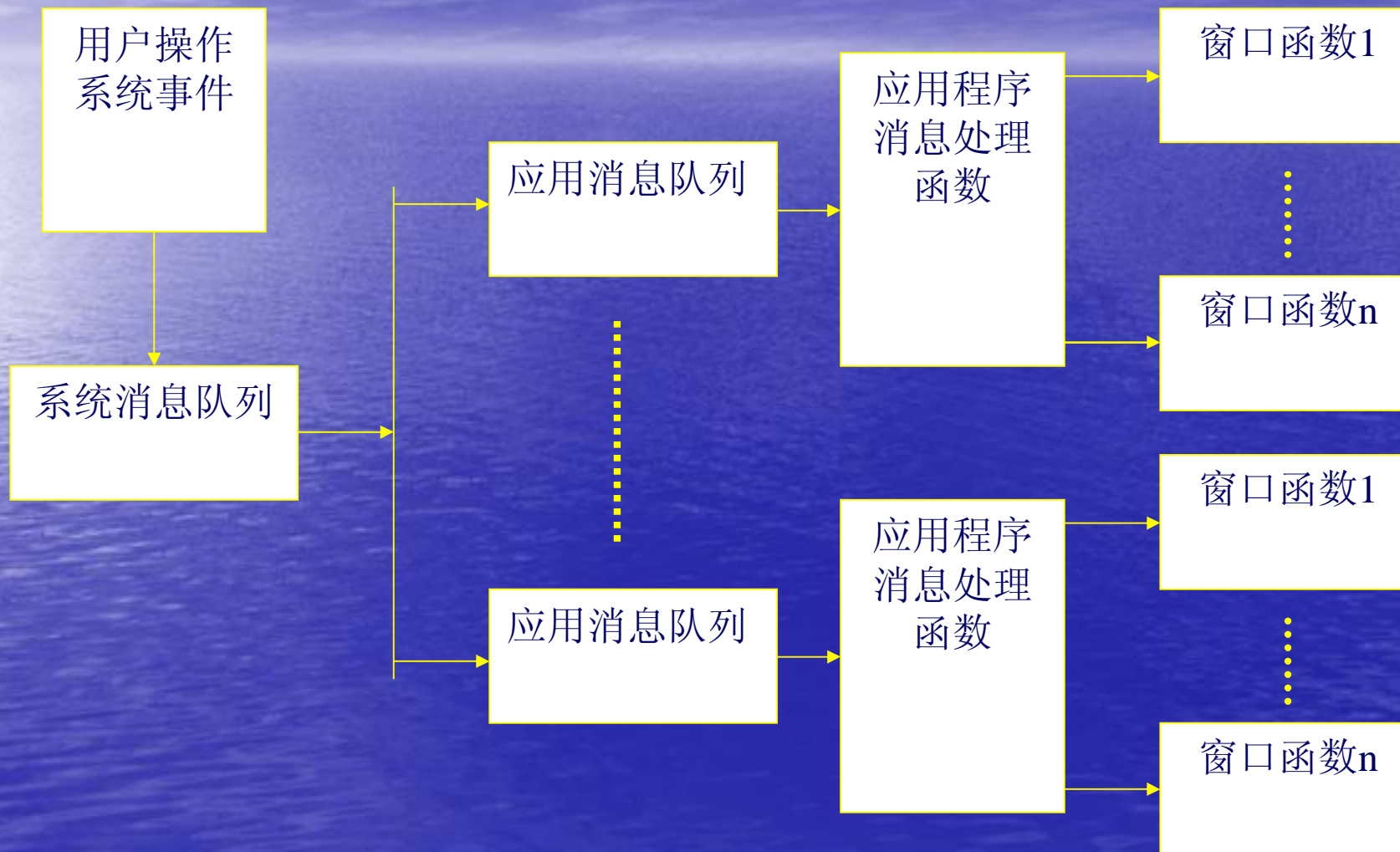


# Windows应用程序设计模式

- 事件驱动



# Windows应用程序设计模式





# Windows应用程序设计模式

## Windows应用程序的开发流程

Windows 应用程序分为程序代码和用户界面资源两部分，两部分通过资源编译器组合为一个完整的EXE文件

将用户界面资源一类的静态数据与程序代码相分离有如下一些优点：

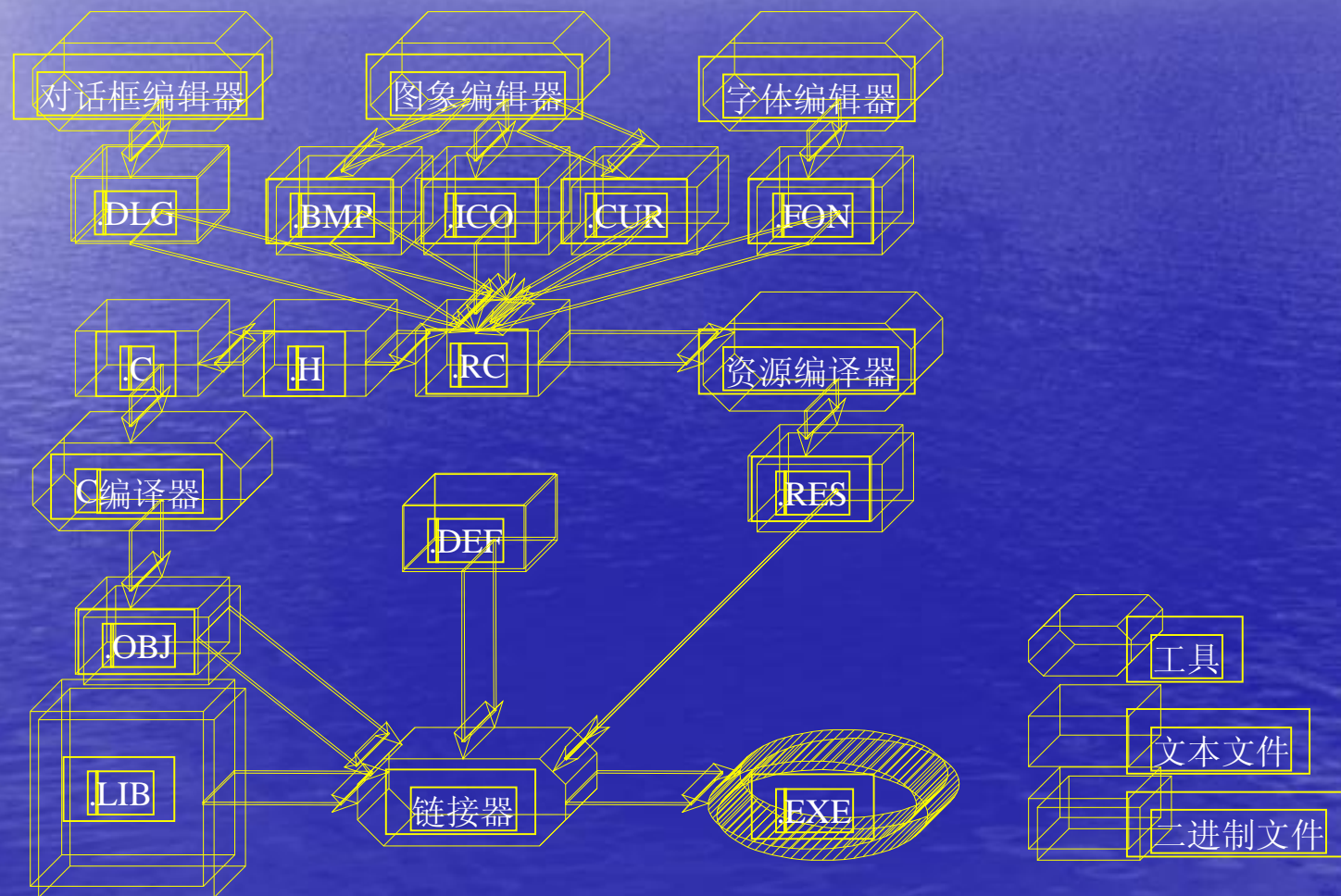
减少内存要求；

划清了程序员与用户界面设计人员的任务分工；

用户界面风格的变化可以不必修改程序代码或只需进行少量的修改。

# Windows应用程序设计模式

## Windows应用程序的开发流程





# Windows应用程序的基本结构

## WinMain函数

WinMain函数是程序的入口点，相当于标准C语言中的main函数

WinMain函数主要由四部分组成：

- 注册窗口类

- 创建窗口

- 显示窗口

- 建立消息循环

- // 程序入口函数
- int PASCAL WinMain(HINSTANCE hInstance,
- HINSTANCE hPrevInstance, LPSTR lpszCmd, int nCmdShow)
- {
- static char AppName[] = "HelloWorld";
- HWND hwnD;
- MSG msg;
- WNDCLASS wndclass;
- // 注册窗口类
- if (!hPrevInstance)
- {
- wndclass.style = CS\_HREDRAW | CS\_VREDRAW;
- wndclass.lpfnWndProc = (WNDPROC)WndProc;
- wndclass.cbClsExtra = 0;
- wndclass.cbWndExtra = 0;
- ● ○ ○ ○ ○ ○ ○ ○ ○
- wndclass.lpszClassName = AppName;
- RegisterClass(&wndclass); }



- // 创建并显示窗口
- hwnd = CreateWindow(AppName,
- "Hello World",
- WS\_OVERLAPPEDWINDOW,
- CW\_USEDEFAULT,
- o o o o o
- hInstance,
- NULL);
- ShowWindow(hwnd, SW\_NORMAL);
- // 消息循环
- while (GetMessage(&msg, NULL, 0, 0))
- {
- TranslateMessage(&msg);
- DispatchMessage(&msg);
- } return msg.wParam;

- // 消息处理的过程函数
- LRESULT FAR PASCAL \_export
- WndProc(HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)
- {
- switch(message)
- {
- case WM\_PAINT :
- {
- char text[] = "Hello World!!";
- PAINTSTRUCT ps;
- BeginPaint(hwnd, &ps);
- TextOut(ps.hdc, 20, 20, text, 13);
- EndPaint(hwnd, &ps);
- break;
- }
- case WM\_DESTROY : {
- PostQuitMessage(0);
- return 0;
- }
- default: return DefWindowProc(hwnd, message, wParam, lParam);
- }



# Windows应用程序的基本结构

## WinMain函数消息循环

Windows并不直接把输入消息发送给应用程序，而是将其送入应用程序的消息队列之中。此外，Windows和其他应用程序也可以将消息指派到应用程序队列中。

应用程序必须读取应用程序队列，检索消息并将它们发送出去，以便适当的窗口函数能够处理它们，负责这一任务的便是消息循环。

```
while(GetMessage(&Msg, NULL, 0,0))  
{  
    TranslateMessage(&Msg);  
    DispatchMessage(&Msg);  
}
```

# Windows应用程序的基本结构

## 窗口函数

窗口函数也称为窗口过程，负责从Windows接收消息，并根据这些消息完成特定的操作

窗口函数是一个回调函数，由Windows系统调用，应用程序并不会直接调用它的窗口函数

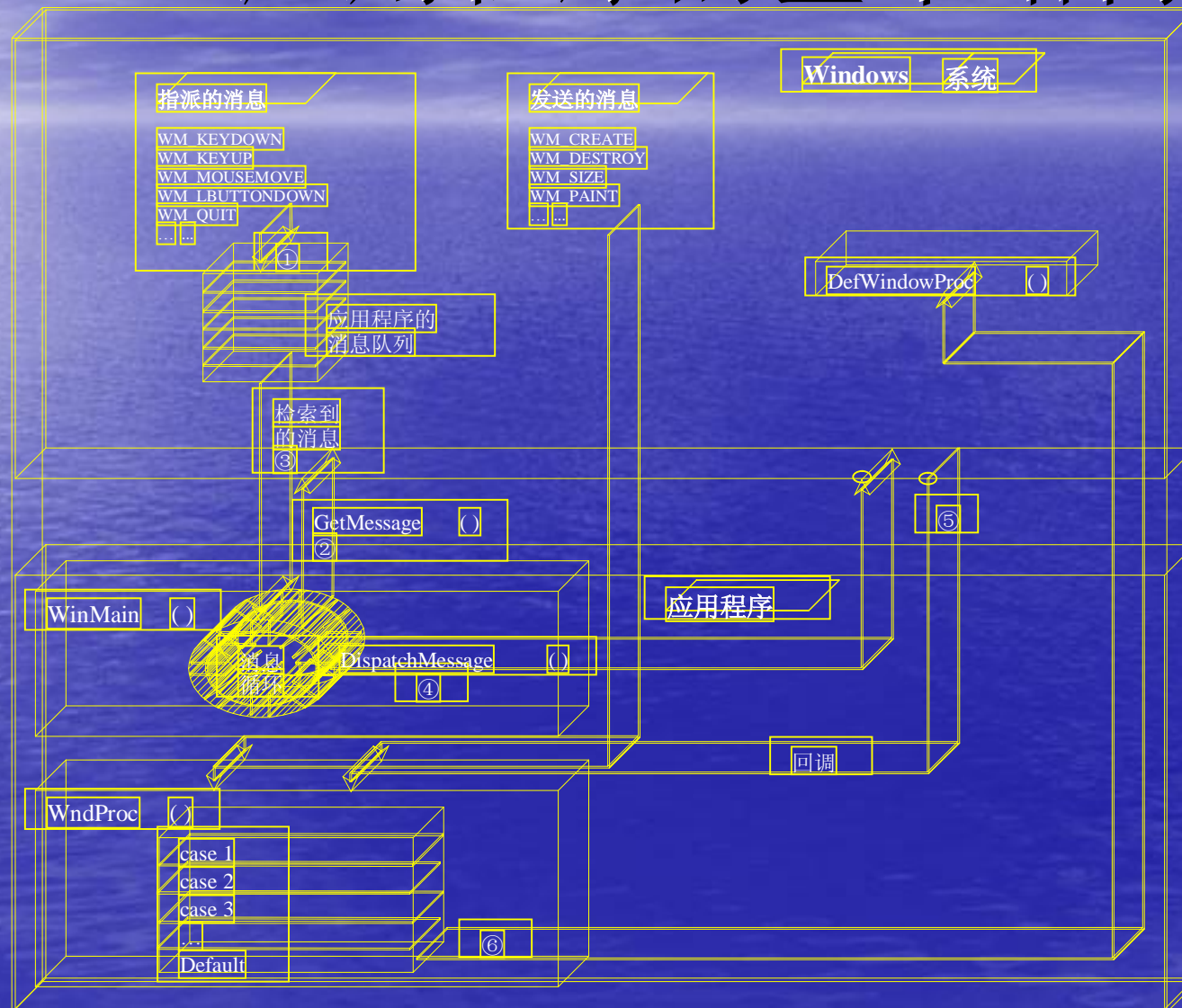
窗口函数的主体是由一系列case语句组成的消息处理程序段

如果窗口函数不处理某些消息，则必须把它们传给DefWindowProc函数

```
LRESULT CALLBACK WindowProc(  
    HWND hWnd,      // handle to window  
    UINT uMsg,      // message identifier  
    WPARAM wParam,  // first message parameter  
    LPARAM lParam   // second message parameter);
```



# Windows应用程序的基本结构



# Windows应用程序的基本结构

当用户关闭窗口时，Windows系统将把WM\_DESTROY消息发送给该窗口的窗口函数，在这种情况下，窗口函数应该使用PostQuitMessage函数将WM\_QUIT消息发送到应用程序队列中，这样可以使GetMessage函数检索到WM\_QUIT消息，从而结束消息循环，退出应用程序。

```
// 消息循环
while (GetMessage(&msg, NULL, 0, 0))
{
    TranslateMessage(&msg);
    DispatchMessage(&msg);
}
return msg.wParam;
}
```



# 异常处理

Windows在系统底层提供了一种称为结构化异常处理SEH的系统机制。利用SEH可以把程序主要的工作同错误处理分离开来，这样的分离，可以使程序员集中精力关注程序要完成的任务，而将可能发生的错误放在后面处理。

异常是在应用程序的正常执行过程中发生的不正常事件。CPU引发的异常称为硬件异常，操作系统和应用程序直接引发的异常，称为软件异常

# 结构化异常处理

SEH是操作系统的一种系统机制，与特定的程序设计语言无关。

应用程序要利用系统提供的SEH机制，则必须借助于特定程序设计语言的相关语法。

因此，SEH不但涉及操作系统，而且与编译器有密切的关系。

结构化异常处理包括异常处理和终止处理两个方面



# 结构化异常处理

## 异常处理

```
__try  
{  
    ...  
}  
__except(exception filter)  
{  
    ...  
}
```

# 结构化异常处理

## 异常处理

异常过滤器返回如下三个异常标识符之一

ECXCEPTION\_EXECUTE\_HANDLER

ECXCEPTION\_CONTINUE\_EXECUTION

ECXCEPTION\_CONTINUE\_SEARCH

## 终止处理

Windows应用程序在运行时通常要分配资源，使用这些资源，然后释放它们。

由于异常改变了控制的流程，因此很容易导致无法释放在产生异常的代码块中分配的资源。

使用终止处理程序可以保证进行这样的清除工作



# 结构化异常处理

## 终止处理

```
__try  
{  
    ...  
}  
__finally  
{  
    ...  
}
```

# 结构化异常处理

有两种情况可能使受保护段不正常地结束：  
在try块中执行了return、goto、break或continue等控制语句  
在try块中发生异常

当一个函数执行失败时，习惯上要返回一些特殊的值来，  
函数的调用者可以检查这些特殊值并采取一种替代的动作  
如果这个调用者是被另一个调用者调用的函数，那么它还需要将它自己的失败代码返回给它的调用者  
这种错误代码的逐层传递会使源程序变得非常难于编写和维护  
采用软件异常则可以解决这些问题



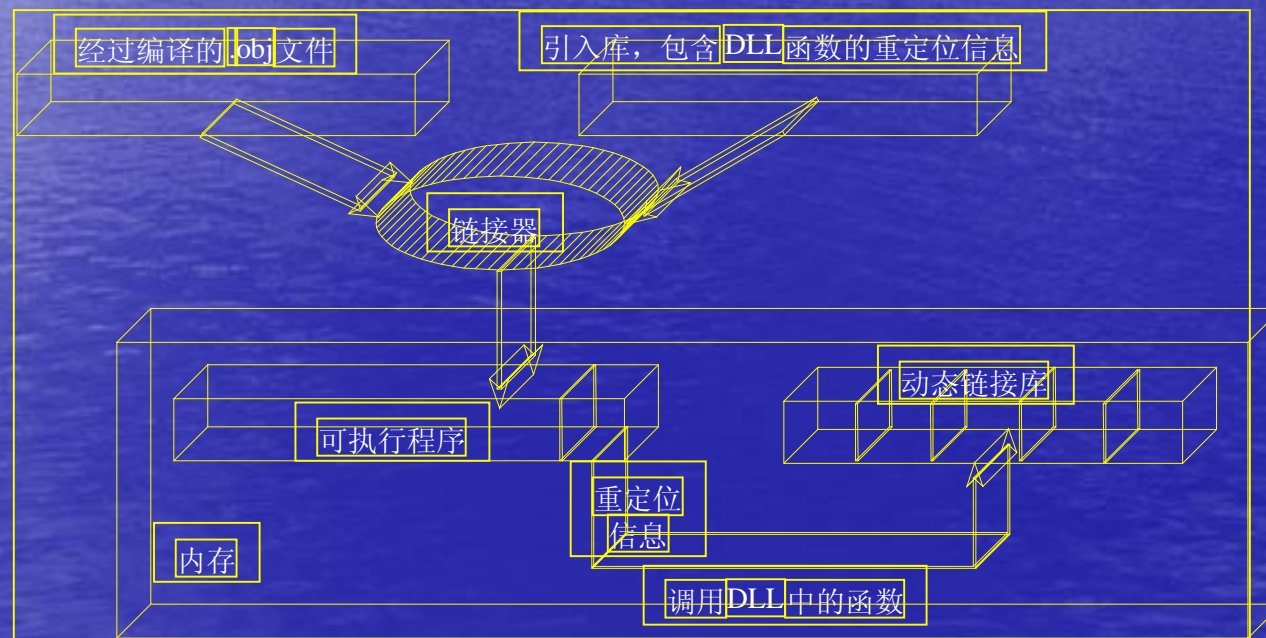
# 动态链接库

1. Windows2000/xp系统平台上提供了一种完全不同的较有效的编程和运行环境,可以将独立的程序模块创建为较小的动态链接库——DLL (Dynamic Linkable Library) 文件,并可对它们单独编译和测试。
2. 在运行时,只有当EXE 程序确实要调用这些DLL 模块的情况下,系统才会将它们装载到内存空间中。这种方式不仅减少了EXE 文件的大小和对内存空间的需求,而且使这些DLL 模块可以同时被多个应用程序用。
3. Windows 自身就将一些主要的系统功能以DLL 模块的形式实现。打开windows 操作系统的system 子目录,就会看到为数众多的带.dll 后缀名的文件(即动态链接库文件)。

# 动态链接库

动态链接库DLL是一个可执行程序模块，模块中包含了可以被其他应用程序或其他DLL共享的程序代码和资源

## 装入时刻动态链接



## DLL到进程地址空间的映射



# 动态链接库

## DLL到进程地址空间的映射

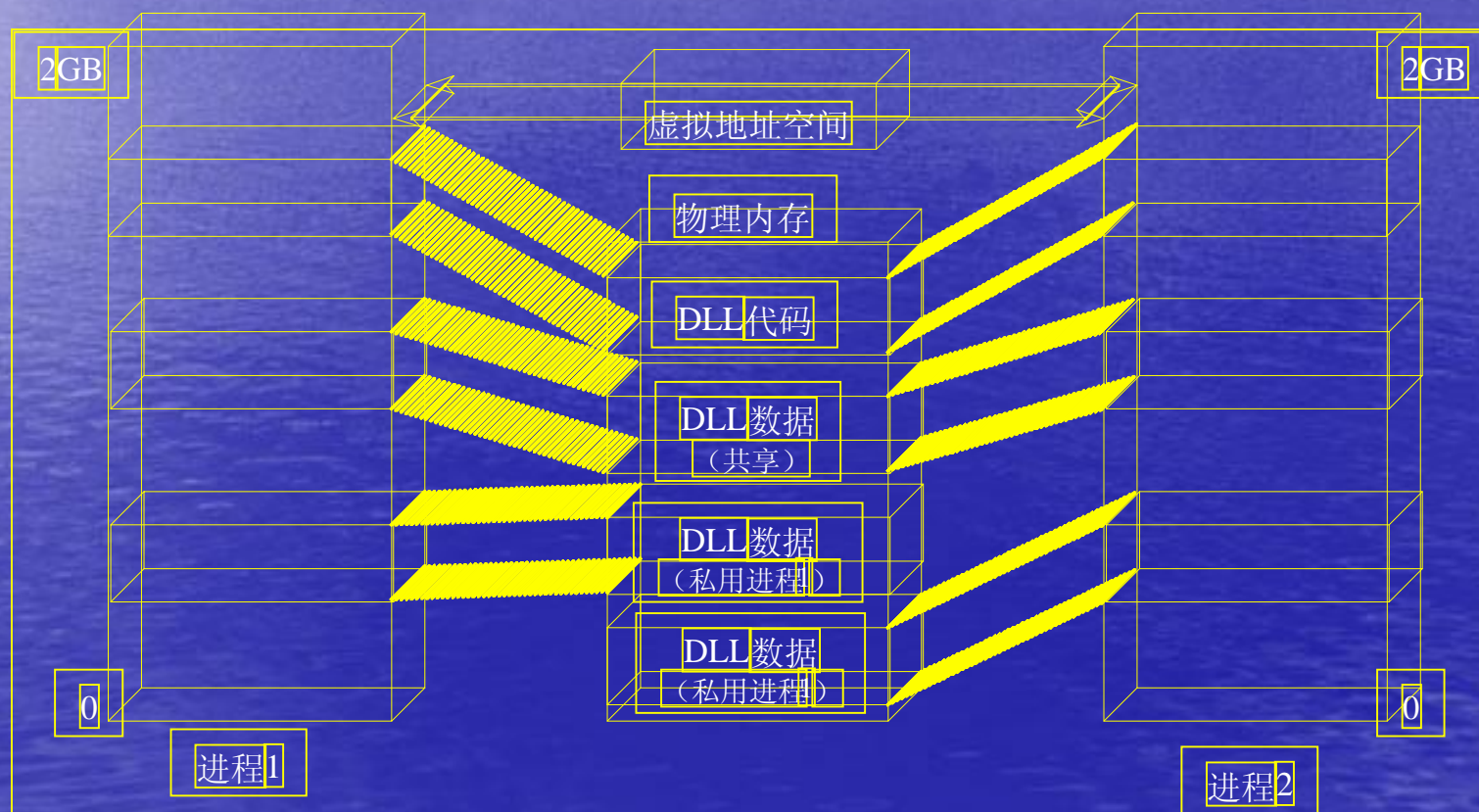
### 运行时刻动态链接

在运行时刻，通过调用LoadLibrary可以使DLL加载到一个进程的地址空间中

为了在运行时刻从DLL中调用一个函数，可以通过调用GetProcAddress获取函数的地址

# 动态链接库

## DLL到进程地址空间的映射





# 动态链接库

## DLL的入口点函数

- 1 DLL没有WinMain函数，不含有消息循环，一般也不获取自己的消息，但是它有自己特殊的入口点函数，入口点函数的缺省名为DllMain
- 2 当进程和线程被初始化或终止时，DllMain函数被Windows系统调用。
- 3 DllMain要做的主要任务是执行进程级或线程级的初始化和清理工作。
- 4 如果不要求DLL初始化，DllMain可以只是一个虚设函数。

# 动态链接库

## DLL的创建和使用

创建DLL文件需要用到源文件（.C）和头文件（.H）。DLL源文件通常包括入口点函数和供应用程序调用的DLL库函数。头文件中含有DLL要导出的所有函数与变量的说明

在应用程序中调用DLL中的函数或访问DLL中的变量时，须告诉编译器要调用的函数或要访问的变量是在DLL中：

```
__declspec(dllimport)  
int Sub(int nPara1, int Para2);
```