第一章　Windows NT 概貌

本书是第一本 Windows NT 编程的实用指南。正如前面所说的，它不全面涉及 Windows NT 的理论方面，除非与编程直接有关的部分。本书提供你尽快开始设计 Windows NT 程序的方法。

尽管如上文所说，在成为一个 Windows NT 程序员之前，从整体上理解 Windows NT 如何运作，它体现什么设计思想以及它如何管理计算机是有必要的。同时知道 Windows NT 与它前代—DOS 和 Windows 的区别也很重要。因此，本章描述 Windows NT 的概貌并讨论它与 DOS 和 Windows 的关系和不同。

如果你以前从未写过 Windows 程序，那么本书的大部分内容对你来说是全新的。如果有耐心、有步骤地进展，当你读完本书后你就会成为很有水平的 Windows NT 程序员。如果你已经编过 Windows 程序，那么你能进展得很快，但要小心，Windows NT 和 Windows 有一些差别，这会影响到你写应用程序。

本书所有例子都用 C 语言写成，并假定你已熟悉 C 语言。C 是 Windows NT 的语言，如果你想对 Windows NT 编程，你必须成为 C 程序员。如果你不懂 C，现在就花点时间学吧。

1.1　Windows NT 是什么

Windows NT 是新一代操作系统，将把 PC 带入下一个世纪。Windows NT 是作为可移植操作系统设计的，能够很容易地在多种不同的硬件平台上运行，包括单处理机和多处理机环境。它可以随着硬件的发展而方便地加以扩展和增强。

你读本书会注意到，可能 Windows NT 最重要的特征在于它是一个完整的 32 位操作系统。随着它发展到完全的 32 位实现，Windows NT 留下许多与原有的 16 位系统有关的问题。

Windows NT 的最主要的设计目标是与其它基于 PC 的操作系统及为它们设计的应用程序兼容。也就是说，Windows NT 是设计成向下兼容大量已有的 PC 应用程序，出于这一目的，Windows NT 包含一些仿真器，可以自动执行为以下操作系统写的应用程序：

- DOS
- Windows(包括 16 位应用程序)
- OS/2
- POSIX(Portable Operating System Interface based on UNIX，基于的可 UNIX 的可移植操作系统界面)

Windows NT 为所要运行的程序类型自动建立合适的运行环境。例如，当用户执行 DOS 程序时，Windows NT 自动建立命令行提示的窗口，程序在其中运行。

Windows NT 的另一个设计考虑就是安全性。Windows NT 提供符合 DoD C2 安全类级的环境。这级安全标准提供口令保护注册，资源访问控制和所有制，以及供审计某一活动的活动日志。并且，内存在被另一个用户使用前加以清除；保护一个程序使用的内存不被另一个程序

占用；由一个程序去使另一个程序运行失败或检查其变量内容是不可能的。

Windows NT 也着意于符合或超过当前的性能标准。在它不可能达到时，如当仿真另一个运行环境时，最多损失 10%的性能。

Windows NT 的另一个重要方面是它可以运行于多 CPU 的计算机上。尽管写此书时几乎没有这样的计算机存在，但它们将来会很普遍。一旦有这样的机器，Windows NT 便可以支持它们。

1.2 Windows NT 如何工作

Windows NT 能够高效地运行于你的计算机、提供对其它操作系统的兼容性并允许扩展，因为它是按照客户/服务模式组织的。你也许知道，存在着几种方式组织操作系统。客户/服务模式与其它方法显著不同在于它组织和执行其代码的方式。

1.2.1 用户模式和核心模式

除了最原始的，所有操作系统都定义系统运行的两个模式。应用程序运行于一个模式，系统代码运行于另一模式。简而言之，采用两个模式的目的是加强操作系统对于计算机的控制，这种方式防止了应用程序不适当地访问系统资源。

当你写一个应用程序时，不能直接与计算机硬件，甚至操作系统层进行交互，而只能与操作系统界面打交道，这样，可以管理应用程序防止它不恰当地调用系统底层或直接访问硬件资源(如端口)。因此，用户模式只有通过核心接口来访问硬件和调用底层服务。

对应的，操作系统核心在核心模式下运行。在核心模式下，一个系统服务可访问所有系统服务和硬件本身。核心模式可全权访问机器并且通常以最高优先级运行。

在最传统的操作系统中，整个操作系统运行于核心模式，应用程序运行于用户模式。对于一个传统的操作系统，这意味着像文件系统、内存管理和 I/O 处理这些都运行于核心模式。

Windows NT 突破了传统的组织方式，体现在以下方面：它将大部分操作系统服务移出核心。因此，Windows NT 核心是很小的，而且从核心中移出的系统服务现在运行于用户模式！正如你将要看到的，将系统服务移出核心就可能对它们更新、修改或加强而不改变操作系统核心。这便得 Windows NT 可扩展、可移植，也易于支持多种不同的操作系统。因为这些服务不再运行于核心模式，而运行用户模式，所以称它们为服务器。

大部分系统服务移出了核心，那么什么保留下来了呢？首先，Windows NT 核心包括用于管理系统中正在执行的任务的调度程序，也包括所有硬件接口代码。最重要的是，它还有与服务器通信的代码。

1.2.2 理解客户/服务模式

由于 Windows NT 将大部分系统服务移出核心变成运行于用户模式的服务器中，你也许会认为 Windows NT 的应用程序就可以自由地与这些服务器直接交互了。然而不是这样的，即使服务运行于用户模式，它们仍是与应用程序全分离加以保护的。保护的方式是客户/服务模式的精华。

客户/服务模式是基于应用程序及其服务器的消息传递。简单地说，一个应用程序(即客

户)访问系统服务(即服务器)唯一途径是向核心传递消息,然后核心把消息传递到适当的服务器上,服务器处理消息并向核心发回响应,由核心再把这个信息返回给应用程序。这样,应用程序就无法与服务器直接通信了,所有通信都是通过核心来完成,而且任何不恰当的访问都会被拒之门外。在这种方式下,操作系统仍旧全控制着计算机。

为更好的理解客户/服务器的关系,我们举一个例子。假如应用程序要打开一个文件。程序向核心发出消息,要求打开一个文件,核心将此消息传递给文件服务器,文件服务器打开文件并获取句柄,再把此句柄传给核心,由核心再将它传回到应用程序中。这一模式可用图1-1来描述。

Figure 1-1
How the client/server model works

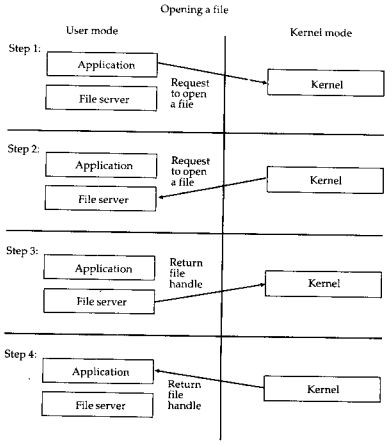

图1-1 客户/服务模式型的工作过程

使用客户/服务器就可在减少核心的同时仍保护和控制对底层系统服务的访问。

客户/服务器相对传统组织方式有几个优点。首先,它可以在不改变核心的情况下,允许维护和更新不同的服务器。简言而之,新的服务器可以随时"进入"而不必触及 Windows NT 的底

层。其次保持一个小的内核有利于内核的可移植性。最后，客户/服务器易于提供对其它操作系统的兼容性。为什么呢？可以这样想：由于服务者运行于用户模式下且不是核心的一部分。修改它们就很容易，进而对应同一种服务可能有多个服务器，每个都模拟一种操作系统，这样，当运行一个为不同的操作系统设计的程序时，它对系统服务的请求就被自动地发送给正确的服务器。正是这种机制使 Windows NT 能支持 DOS、OS/2、POSIX 和 Windows 程序。

1.3 理解进程和线索

你肯定知道，Windows NT 是一个多任务操作系统，它能够同时运行两个以上的程序。在仅有一个处理器的系统中，程序共享 CPU 而不是真正技术上讲的同时运行，虽然计算机的速度极快使它们看上去就像同时运行。而 Windows NT 能够在有两个以上 CPU 的计算机上运行。这时，程序可能真正地同时执行。既然你不会总是知道程序在哪台计算机上运行，最好还是想像运行总是真正同时的。

与其它操作系统不同，Windows NT 支持两种多任务形式：基于进程的和基于线索的。一个进程是一个正在执行的程序。由于 Windows NT 可以处理多进程，多个程序可以同时运行。这是传统的多任务形式，你可能已很熟悉。

Windows NT 的第二种多任务形式是基于线索的。一个线索是一个可分离的执行代码片断。这个名称来源于"执行线索"的概念。每个进程至少有一个线索，而 Windows NT 进程可有多个线索。

因为 Windows NT 能处理多线索而且一个进程能有一个以上的线索，你会想到有可能使一个进程的两个或多个片断同时执行。是这样的，使用 Windows NT，同时处理多个程序和一个程序的多个片断是可能的。

1.4 Windows NT 基于调用的接口

如果你有 DOS 编程背景，那么你知道 DOS 是通过许多软中断作为接口的。比如，标准 DOS 中断是 0x21。虽然用软中断访问系统服务已被广泛接受（在给定的有限范围的 DOS 操作系统中），但对于像 Windows NT 这样的全功能多任务操作系统来说，这种接口方式是远远不够的。Windows NT 和 Windows 一样，使用基于调用的接口来访问操作系统。

Windows NT 基于调用的接口使用一组丰富的系统定义函数来访问操作系统功能，合称应用程序编程接口，简称 API。API 包含有几百个函数，由应用程序调用来与 Windows NT 通信。这些函数提供所有必要的操作系统功能，诸如内存分配，屏幕输出和窗口创建等。

1.5 动态链接库（DLLs）

API 由几百个函数组成，你也许会设想为 Windows NT 编译的程序会链接上大量的代码，造成每个程序中都有大量重复代码。但事实不是如此，相反 Windows NT 函数包含于动态链接库（简称 DLL）中，当程序执行时可以访问它们。下面解释动态链接库的工作方式。

Windows NT API 函数以可重定位格式存放在 DLL 中。在编译阶段，程序中调用 API 函

数时连接程序并不把那个函数的代码加入到程序的可执行版本中，而是加进此函数的装入指令，DLL 的位置和函数名。当程序执行时，必要的 API 代码被 Windows NT 装入程序装入。这样应用程序就不需包含实在的 API 代码存在磁盘上，而仅当这些程序被装入内存执行时才装入这些代码。

动态链接库有许多重要的优点。首先，由于几乎所有程序都会使用 API 函数，DLL 避免了大量重复目标代码造成的磁盘空间的浪费，这些重复代码若真正地加入到每个程序的可执行文件中数量极大。其次，对 Windows NT 的更新和增强能够由修改动态链接库来完成，而已有的应用程序不必重新编译。第三，动态链接库使得模拟其它操作系统更加容易。

1.6 Windows 与 Windows NT 对比

尽管 Windows NT 是 Windows 系列产品序列中的新产品之一，(Windows 最初发布于 1985 年)，但它在操作系统设计中前进了一大步。而从应用程序员的角度看，编程方法是相同的。

一个好消息是，如果你熟悉 Windows 那么就能毫无问题地使用 Windows NT 或对它编程。从用户角度看，Windows NT 看上去就像 Windows 一样，有程序管理器、文件管理器和附件等，并多少以相同的方式工作。从本书的目的来讲，更重要的是你几乎可以像对 Windows 编程一样来编写 Windows NT 程序。Windows NT 保留了 Windows 原有的函数名字空间，并通过加入新函数来加入新的功能。虽然 Windows 和 Windows NT 有些不同，但这些不同是很容易接受的，并且，原有的 Windows 程序在 Windows NT 下运行得很好，所以你不必马上去处理你所有的应用程序。

对用户而言，更大程度上对程序员而言，Windows NT 和 Windows 看上去很相同，但仍有一些不同之处，其中最重要的差别总结如下。

1.6.1 从用户角度看到的不同

从用户的观点看，Windows NT 充实了许多功能，包括透明运行 DOS 程序的能力。当你运行 DOS 程序时，命令行提示的窗口界面自动创建起来。进一步讲，这一界面又完全是 Windows NT 图形界面的组成部分。

Windows NT 支持安全性，所以它有几个与安全有关的功能。首先它有用户帐户，用于维护存取权限的不同。你必须向 Windows NT 注册进入一个用户帐户，按下 CTRL－ALT－DEL 不再重启动计算机，而是只简单地进入注册屏幕。

Windows NT 还包括许多 Windows 没有支持的附件及管理工具。

1.6.2 从程序员角度看到的不同

从程序员的观点看，Windows 与 Windows NT 主要有两点不同。第一，Windows NT 支持全 32 位地址并使用虚拟内存。Windows 使用 16 位段寻址模式。对于许多应用程序这个变化几乎没有影响。而对于某些其它程序影响很大。坦率地说，这一转换不会没有麻烦，但你会发现 Windows NT 32 位内存模式会大大简化编程。

第二个不同是有关多任务实现的方式。Windows 采用非抢占式任务切换方式。这意味着，

要运行其它任务必须手工地将控制返回到调度程序。换句话说，Windows 程序保持对 CPU 的控制，直至它决定放弃控制为止。因而，一个病态的程序会使 CPU 出现极端情况。相反 Windows NT 采用抢占式的、基于时间片的任务机制。Windows NT 自动抢占任务，CPU 于是被分配给下一个任务（如果有的话）。抢占式多任务机制是一个更为高级的方法，它使得操作系统完全控制了任务，并防止了一个任务独占系统的情况出现。大部分程序员认为发展到抢占式多任务机制是一个进步。

除了以上两点主要的差别，Windows NT 与 Windows 还有其它一些较不显著的不同，以下讨论之。

1）输入队列

Windows NT 与 Windows 的另一个不同之处在于输入队列（输入队列保存一些消息，如键按下或鼠标动作等，直到它们被发送到程序中。）在 Windows 中，系统中所有正在运行的任务只有一个输入队列。而 Windows NT 为每个线索提供其自己的输入队列。每个线索拥有自己的队列的优点是，任何进程都不能因迟缓响应消息而降低系统效率。

尽管多输入队列是个重要的变动，但这不会直接影响你对 Windows NT 编程。

2）动态链接库（DLL）

尽管 Windows 和 Windows NT 都采用基于 DLL 库的 API 来访问操作系统，但实际的 DLL 的内容是不同的（一般而言，这些不同从编程角度看没有关系）。在 Windows 中，DLL 实实在在地包含 API 函数，当可执行程序运行时它们被链接上去。而在 Windows NT 中，DLL 中包含短小的代码片断，称为 Stub。它将应用程序的函数调用转换成消息，再传递给核心，核心再把这个消息传递给恰当的服务器，服务器按消息的要求完成必要的工作后，把结果返回给核心，最后再通过 Stub 传回给应用程序。虽然在这一机制上 Windows NT 与 Windows 有很大不同，但这不影响对 Windows NT 编程。

3）控制台（Console）

过去，基于正文（非窗口）的应用程序要在 Windows 下运行相当不方便。而 Windows NT 支持一类特殊的窗口，叫控制台。一个控制台窗口提供一个标准的命令提示界面，但除了全基于正文外，一个控制台可像其它窗口一样地操纵和动作。基于正文的控制台环境下运行，而且便于你做短小的一次性的实用程序。可能更重要的是，在 Windows NT 中加入控制台最终承认某些基于正文的应用程序仍是有用的；现在它们可以在全窗口环境中管理起来。总之，加入控制窗口完备了 Windows 应用程序环境。

4）平面寻址（Flat Addressing）

Windows NT 应用程序拥有 4GB 的虚拟内存作为运行空间。而且，这一地址空间是平面的。与 Windows，DOS 和其它 8086－系列使用分段内存的操作系统不同，Windows NT 认为内存是线性的。由于 Windows NT 采用虚拟内存，每个应用程序就有了大大多于它可能合理占用的内存空间。虽然平面内存的变化对于程序员来说是透明的，但这确实减轻了许多过程单调烦人的处理分段的工作。

5）消息和参数类型的变化

由于 Windows NT 转换为 32 位寻址，就要对一些发送给 Windows NT 程序的消息进行不同地组织，这不同于发送给 Windows 程序的情况。用于声明窗口函数的参数类型也由于 32 位寻址而改变了。具体的变化将在本书后面用到时加以讨论。

1.7 需要配备什么软件

在写本书时，对 Windows NT 程序只有使用 Mincrosoft 的 Windows NT Developers kit(正式称为 WIN32 Software Developers kit for Windows NT，我们不会用这么长的名字叫它！)这个开发工具中有可编译本书中所有程序的 C/C++编译器。

当你读此书时，可能会有其它编译器生成 Windows NT 代码。本书的程序可使用任何 Windows NT 兼容的编译器编译。如果使用不同的编译器，特定的编译选项和 make 文件指令会有所不同。

本书所有例子都用标准 C 写成，以使于可以更普遍地被编译器编译。但最新近的编译器都支持 C++扩展，你没有理由不用 C++来写你自己的程序。

1.8 术语

为避免混淆，以后本书将使用以下术语。Windows 的 16 位版本称为 16 位 Windows。Windows NT 就是这样确切地称呼。当讨论基本环境或区别无关紧要时，就用 Windows。

1.9 转换注释

由于许多读者要将 16 位 Windows 应用程序转换到 Windows NT 上，转换注释将在合适的地方加入。尽管把 16 位应用程序转换到 Windows NT 基本上不困难，但要强调几点重要的不同之处。

第二章　Windows NT 编程概貌

本章介绍 Windows NT 编程，主要有两个目的。第一，讨论程序如何用 Windows NT 交互及每个 Windows NT 应用程序必须遵循的要求。第二，开发一个骨架应用程序作为所有其它 Windows NT 程序的基础。你将看到，所有 Windows NT 程序都有一些共同点，这些共同的特性会体现在骨架应用程序中。

正如前一章提到的，虽然原则上相似，但 16 位 Windows（如 Windows3.1）和 Windows NT 间仍存在着重要的不同之处，出现时将加以讨论。如果要引入原有的程序请特别注意这些不同点。

注释：如果你已经知道怎样写 16 位 Windows 程序，那么本章和下面几章包括的基本 Windows 编程概念和技术你都已熟悉。（实际上，Windows 和 Windows NT 编程在表面上几乎是相同的），但你至少浏览一下这些章节，因为其中有你必须知道的 16 位 Windows 与 Windows NT 的不同点。

2.1　Windows NT 编程透析

Windows NT（和一般意义的 Windows）的目标是能够让对系统有基本了解的用户坐下来运行几乎任何应用程序，而不必提前加以培训。理论上讲，会运行一个 Windows 程序，就会运行所有的。当然，实际上为了更有效地工作，通常要进行某些培训。但至少是限制在告诉用户程序怎么做，而不是用户怎样同程序交互。事实上，许多 Windows 应用程序代码只是在支持用户界面。

关于这一点，你必须清楚并非每一个运行在 Windows NT 下的程序都必然使用户看到 Windows 风格的界面。只有那些利用了 Windows 写的程序在观感上才像 Windows 程序。当你在改变 Windows 设计原则时，你必须有很好的理由，因为程序的用户很有可能被搞糊涂。诚实地说，如果你为 Windows NT 写应用程序，就应该遵循和接受 Windows 编程原则。

Windows NT 是面向图形的，即它提供图形用户接口（GUI）。尽管图形硬件和视频模式各种各样，但它们的差别都由 Windows 处理了。这意味着在很大程度上，程序不必担心正在使用的图形硬件和视频模式。

下面来看一看 Windows NT 的一些重要特性。

2.1.1　桌面模型

几乎没有例外，基于 Windows 的界面提供的屏幕等同于桌面。在桌面上会有几张不同的纸张，一张摞在另一张上，通常上面的盖住下面的，而下面的又有一部分露出来可见。Windows NT 中与桌面等同的是屏幕，与纸张等同的是屏幕上的窗口。在桌面上，你能把纸张移来移去，把下面的拿到上面来，或调整另一张纸可以看到多少。Windows NT 对于窗口有统一类型的操作。选中一个窗口，便使之成为当前窗口，它也就位于其它所有窗口的上面了。用户可以扩大

或缩小一个窗口，或在屏幕上移动这个窗口。总之，Windows 可以让你像处理桌面一样控制屏幕。

2.1.2 鼠标

在 Windows NT 中可以用鼠标来处理几乎所有的控制、选择和作图操作。当然，说可以用鼠标是不够的，Windows NT 也可以用键盘。虽然很可能应用程序忽略鼠标。但这就违反了 Windows 设计的基本原则。

2.1.3 图标和图形图像

Windows NT 允许(但不要求)使用图标和位图图像。使用图标和图像的理论在于一句老话：一图胜千言。

一个图标是一个小的图像符号，用来表示某种功能或可以用鼠标双点激活的应用程序。图像通常用来向用户传递某种信息。

2.1.4 菜单和对话框

除了标准窗口外，Windows NT 还提供特殊用途的窗口，最普遍的是菜单和对话柜。菜单，正如你所知，是一种用户可以作出选择的特殊窗口。

对话框也是一种特殊窗口，但可以进行比菜单选择更加复杂的用户交互。比如，应用程序可以用对话输入文件名。几乎没有例外，非菜单的输入一定是由对话框完成的。

2.2 Windows NT 和应用程序如何交互

当你为许多操作系统写程序时，是程序首先发起同操作系统交互。例如，在一个 DOS 程序中，是程序要求进行诸如输入和输出的操作，换而言之，传统方式写的程序是在调用操作系统，而 Windows NT 在大部分情况下的以相反的方式工作：Windows NT 调用你的程序。其处理工作是这样的：Windows 程序要等到 Windows 发来的一个消息，消息通过 Windows 调用的一个特殊程序传送到你的程序。一旦消息被接收，应用程序就可以进行适当的动作。当应用程序响应一个消息而调用一个或多个 Windows NT API 函数时，仍是 Windows NT 来发起这个动作。可以说，基于消息的与 Windows NT 交互的方式标志着 Windows NT 应用程序的一般形式。

Windows NT 可能发送许多不同类型的消息给应用程序。例如，每当鼠标在属于应用程序的窗口中撤动时，一个"鼠标按动"的消息就发送到这个程序；每当属于程序的窗口必须刷新时也有一类消息发送过去；每当程序接受了输入焦点而用户按下一个键时，另一类消息又发送过去，等等。要牢牢记住一个事实，就是对于应用程序，消息的到达是随机的。这也是 Windows NT 程序类似中断驱动程序的原因。你不会预料下一条消息是什么。

2.3 Windows NT API

总的来讲，Windows 环境是通过基于调用的接口来访问的，这个接口叫 API(Application Program Interface)。几百个 API 函数给出了所有由 Windows NT 完成的系统服务。API 有一

个子集叫 GDI(Graphics Device Interface,图形设备接口),这是 Windows 提供设备无关图形支持的部分。正是 GDI 函数和 Windows 函数使 Windows 应用程序可以运行在各种各样的硬件配置上。

API 的大部分函数由原来的 16 位 Windows 版本支持,也与 Windows NT 兼容。的确,这些函数大部分用相同的名字调用并以相同的方式使用。但即使在用途和原理上很相似,两种 API 仍有区别。因为正如第一章提到的,Windows NT 支持全 32 位寻址,而 16 位 Windows 只支持 16 位、分段内存模式。这个区别使 API 函数扩展为接受 32 位参数和返回 32 位值。一小部分 API 函数必须有变化以兼容 32 位体系。如果你刚接触 Windows 编程,这些变化不会明显地有影响。但如果你要把代码从 16 位 Windows 向 Windows NT 移植,那么就要仔细研究传给每个 API 函数的参数了。

由于 Windows NT 支持全 32 位寻址,使用 32 位长的整数就有意义了。这就是说 int 和 unsigned 类型将是 32 位长,而不是 16 位 Windows 中的 16 位长了。如果要用 16 位整数,就要声明为 short 类型。(很快会看到,Windows NT 提供可移植的 typedef 名字。)这意味着从 16 位环境移植代码,要检查你使用的整数类型和可能引起的副作用。

另一个 32 位寻址的结果是指针不用再声明为 near 或 far。任何指针都可以访问内存任一部分。在 Windows NT 中,far 和 near 定义没有意义。这就是说你可以在向 Windows NT 移植时可以保留程序中的 far 和 near,但它们都没有作用。

2.4 窗口的组成要素

在讲 Windows NT 编程的特殊方面之前,要定义几个重要术语。

图 2—1 是一个标准窗口,图上标出了其每个元素。

所有窗口都有一个边界,限定窗口范围并用于缩放窗口。在窗口顶部有几个元素。最左边是系统菜单图标,点中这个方框可以显示出系统菜单。系统菜单右边是窗口标题。最右边是极大化和极小化方框,用户区是窗口中用户程序工作的部分。多数窗口都有横向和纵向滚动条,用来翻看窗口中的内容。

2.5 Windows NT 应用程序的一些基本概念

在开发 Windows NT 应用程序骨架之前,先来讨论一下 Windows NT 程序的基本概念。

2.5.1 WinMain()

所有 Windows NT 程序从调用 WinMain()开始执行。(Windows 程序没有 main()函数)WinMain()有一些特殊性质与程序中其它函数不同。首先,它必须用 WINAPI 调用约定来声明。缺省时,C 或 C++程序的函数使用 C 调用约定。在编译一个函数时,可以使用不同的调用约定。例如,通常也使用 Pascal 调用约定。出于技术原因,Windows NT 调用 WinMain()的调用约定是 WINAPI。WinMain()的返回类型为 int。

16 位转换注释　在 16 位 Windows 中,对 WinMain()的调用约定是 FAR PASCAL。在向 Windows NT 移植时,应更为 WINAPI。

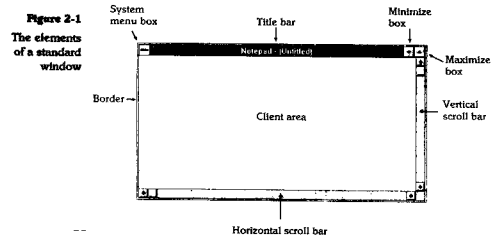

图 2－1　标准窗口的元素

2.5.2　窗口函数

所有 Windows NT 程序中必须有一个特殊函数，它不被你的程序调用，而是由 Windows NT 来调用。这个函数一般称为窗口函数。当需要向你的程序传递消息时，Windows NT 就调用窗口函数。Windows NT 就是通过这个函数与应用程序通信的。窗口函数从其参数中接收消息。所有 Windows 函数必须声明为返回类型 LRESULT CALLBACK。LRESULT 类型是由 typedef 定义的 LONG 类型的别称，CALLBACK 调用约定是由 Windows NT 所调用的函数使用的。在 Windows 术语中，任何由 Windows 调用的函数都叫回调函数(Callback Function)。

16 位转换注释　16 位 Windows 代码将窗口函数定义为 FAR PASCAL。尽管 Windows NT 出于可移植性考虑允许这样定义，但你应在所有的新的 Windows NT 代码中使用 CALLBACK。

除了接收 Windows NT 发送的消息外，窗口函数必须发起消息所指示的动作。通常，窗口函数体中有一个 switch 语句，将特定的响应动作与程序收到的消息联系起来。程序没有必要响应每一个 Windows NT 发出的消息。对于程序不关心的消息，可以采用 Windows NT 提供的缺省处理。由于 Windows NT 可产生 100 多条不同的消息，通常大部分消息都由 Windows NT 处理了，而不用你的程序干预。

所有消息都是 32 位整数值，而且与消息要求的附加信息相连。

2.5.3　窗口类

当 Windows 程序开始执行时，需要定义并注册一个窗口类。(在此，类这个词与 C++中的含义不同，它表示风格类型。)注册窗口类时，就告诉了 Windows NT 窗口的形式和功能。但

是注册窗口类并不生成窗口，创建一个窗口要再完成另外几步。

2.5.4 消息循环

前面解释过，Windows NT 通过发送消息与应用程序通信。所有的 Windows NT 应用程序必须在 WinMain()函数里建立一个消息循环。这个循环从应用程序消息队列中读取消息，再将消息发回 Windows NT，后者再把消息作为参数来调用应用程序的窗口函数。这好像是个过于复杂的消息传递方式，但所有的 Windows 程序都这样动作。(采用这个方式的原因之一是可以将控制返回给 Windows NT，以便调度程序分配 CPU 时间，而不是等待程序时间片结束。)

2.5.5 Windows 数据类型

很快就可以看到，Windows NT 程序并不扩展的 C/C++ 数据类型，而是在 windows.h 及其相关文件中用 typedef 定义大部分数据类型。这个文件由 Microsoft(或其它提供编译器的公司)提供，必须包含在所有 Windows 程序中。常用的类型有 HANDLE、HWND、BYTE、WORD、DWORD、UINT、LONG、BOOL、LPSTR 和 LPCSTR。HANDLE 是一个 32 位整数句柄。Windows 中使用多种句柄类型，它们都和 HNANDLE 同样大小，一个句柄就是一个标识资源的值。例如，HWND 是作为窗口句柄的 32 位整数。所有的句柄类型都以 H 开头。BYTE 是 8 位的 unsigned char，WORD 是 16 位 unsigned int，LONG 是 long 的别名，BOOL 是整数，用来标识一个值是真或是假。LPSTR 是指向字符串的指针，LPCSTR 是指向字符串的 const 指针。

除了以上介绍的基本类型，Windows NT 还定义了一些结构。骨架程序需要的两个是 MSG 和 WNDCLASS。MSG 结构中包含 Windows NT 消息，WNDCLASS 定义窗口类。本章后面还要讨论这些结构。

16 位转换注释　Windows NT 程序时，UINT 是 32 位无符号整数。如果要编译成 16 位 Windows 代码，它变为 16 位无符号整数。

2.6 一个 Windows NT 骨架程序

既然必要的背景知识已经说明，就要准备开发一个极小的 Windows NT 应用程序了。所有 Windows 程序都有某些共同的东西，这一节就开发一个 Windows NT 骨架程序，其中包含必要的 Windows 特征。最小的 DOS 程序约有五行，最小的 Windows 程序大约 50 行。在开发 Windows 程序时，骨架程序常常被采用。

一个最小的 Windows 程序包含两个函数：WinMain()和窗口函数。WinMain()函数必须完成以下几步：

- 定义一个窗口类
- 向 Windows NT 注册这个类
- 创建这个类的窗口
- 显示这个窗口
- 开始进行消息循环

窗口函数必须响应所有有关的消息。由于骨架程序仅仅显示其窗口，它只需响应一个消

息，即告知应用程序用户已经结束程序的消息。

进行具体讨论之前，先看一下下面的这个最小的 Windows NT 应用程序骨架。它创建一个带有标题的窗口。窗口还带有系统菜单，因而可以极大化、极小化、移动、放缩及关闭。它还带有标准的极大化和极小化图标。

```
/* A minimal Windows NT skeleton */
/* The following definition causes stricter type checking.
This is optional, but suggested because it will help
catch potential type mismatch errors, especially
when porting from 16 bit Windows. */
#define STRICT
#include <windows.h>

LRESULT CALLBACK WindowFunc(HWND,UINT,WPARAM,LPARAM);

Char szWinName[]="MyWin"; /* name of window class */
int WINAPI WinMain(HINSTANCE hThisInst,HINSTANCE hPrevInst,
                   LPSTR lpszArgs, int nWinMode)
{
  HWND hwnd;
  MSG msg;
  WNDCLASS wcl;

  /* Define a window class */
  wcl.hInstance=hThisInst; /* handle to this instance */
  wcl.lpszClassName=szWinName; /* window class name */
  wcl.lpfnWndProc=WindowFunc; /* window function */
  wcl.style=0; /* default style */

  wcl.hIcon=LoadIcon(NULL,IDI_APPLICATION); /* icon style */
  wcl.hCursor=LoadCursor(NULL,IDC_ARROW); /* cursor style */
  wcl.lpszMenuName=0; /* no menu */

  wcl.cbClsExtra=0; /* no extra */
  wcl.cbWndExtra=0; /* information needed */

  /* Make the window light gray. */
  wcl.hbrBackground=GetStockObject(LTGRAY_BRUSH);

  /* Register the window class. */
  if(!RegisterClass(&wcl)) return 0;

  /* Now that a window Class has been registered, a window
    can be created. */
  hwnd = CreateWindow(
    szWinName, /* name of window class */
```

```
    "Windows NT Skeleton",  /* title */
    WS_OVERLAPPEDWINDOW,  /* window style - normal */
    CW_USEDEFAULT,  /* X coordinate - let Windows decide */
    CW_USEDEFAULT,  /* Y coordinate - let Windows decide */
    CW_USEDEFAULT,  /* width - let Windows decide */
    CW_USEDEFAULT,  /* height - let Windows decide */
      NULL,  /* handle of parent window - there isn't one */
      NULL,  /* no menu */
      hThisInst,  /* handle of this instance of the program */
      NULL  /* no additional arguments */
    );

    /* Display the window. */
  ShowWindow(hwnd,nWinMode);
  UpdateWindow(hwnd);

  /* Create the message loop. */
  while(GetMessage(&msg,NULL,0,0))
  {
    TranslateMessage(&msg);  /* allow use of keyboard */
    DispatchMessage(&msg);  /* return control to Windows */
  }
  return msg.wParam;
}

/* This function is called by Windows NT and is passed
  message from the message queue.
*/
LRESULT CALLBACK WindowFunc(HWND hwnd,UINT message,WPARAM wParam,
                            LPARAM lParam)
{
  switch(message)
  {
    case WM_DESTROY:  /* terminate the program */
      PostQuitMessage(0);
      break;
    default:
      /* Let Windows NT process any message not specified in
        the preceding switch statement. */
      return DefWindowProc(hwnd,message,wParam,lParam);
  }
  return 0;
}
```

我们一步一步地看一下这个程序。

首先，所有 Windows 程序都要包含头件 windows.h。这个文件中有 API 函数原型和 Windows NT 使用的各种类型和定义。例如，windows.h 中定义了 HWND 和 WNDCLASS 类型。#define 前的注释指出，定义了 STRICT 宏使编译器进行更强的类型检查。（必须在包含 windows.h 前定义。）在刚开始开发 Windows NT 程序或移植原有的 16 位 Windows 应用程序时，最好定义它。如果你不使用 Microsoft 的编译器，可能没有这一功能。如果没有，那么就简单地保留这个语句。

程序使用的窗口函数叫 WindowsFunc()，它被声明为回调函数，因为这是 Windows NT 与程序通信的函数。

程序从 WinMain()开始执行。Windows 传给 WinMain()四个参数。hThis Inst 是程序当前实例的句柄。记住，Windows NT 是多任务系统，在同一时刻可能有多个实例在运行。hPreInst 永远是 NULL（在 16 位 Windows 程序中，如果当前程序的其它实例运行，hPrevInst 就非零，但这不再适用于 Windows NT）。lpszArgs 是一个指向字符串的指针，这个字符串包含应用程序启动时的命令行参数。nWinMode 的值决定当程序启动时是显示窗口还是以图标形式显示。

在函数中，建立了三个变量。hwnd 变量包含了标识程序窗口的值。msg 结构变量包含窗口消息。wcl 结构变量用来定义窗口类。

16 位转换注释　上面提到，hPrev Inst 在 Windows NT 中永远是 NULL，原因在于 16 位 Windows 和 Windows NT 的根本区别。在 16 位 Windows 中，一个程序的多个实例共享窗口类和其它数据。因此，知道系统中是否有自己的其它版本在运行对于程序来说很重要。但在 Windows NT 中，每个进程都是分立的，不自动共享窗口类或类似数据。Windows NT 中保留 hPrevInst 的唯一原因就是兼容性。

2.6.1　定义窗口类

WinMain()的头两个动作就是定义窗口类和注册之。通过填充 WNDCLASS 结构的域定义窗口类，下面是它的各个域：

```
UINT style;/* type of window */
WNDPROC lpfnWndProc; /* address to window func */
int cbClsExtra; /* extra class info */
int cbwndExtra; /* extra windows info */
HINSTANCE hInstance; /* handle of this instance */
HICON hIcon; /* handle of minimized icon */
HCURSOR hCursor; /* handle of mouse Cursor */
HBRUSH hbrBackground; /* background color */
LPCSTR lpsz MenuName; /* name of main menu */
LPSTR lpsz Class Name; /* name of window class */
```

看过程序知道，hInstance 域被赋以 hThis Inst 标识的当前实例句柄。LpszClassName 指向窗口类名字“MyWin”。窗口函数地址赋给了 lpfnWndProc。这里使用缺省风格，不需要额外信息。

16 位转换注释　在 16 位 Windows 中，每个窗口类必须只注册一次。因此当程序的其它实例运行时，当前程序不能定义和注册窗口类。为避免出现这种情况，要检查 hPrevInst 的值。如

果它非零，那么窗口类已被其它实例注册，否则窗口类就被定义并注册。Windows NT 不再需要检查 hPrevInst，本书的例子也有再包含检查 hPrevInst 的代码，但为了向下兼容 16 位 Windows，你可以写入这些代码，不会有影响。

所有 Windows 程序都要为鼠光标和极小化图标定义一个缺省形状。应用程序可以定义其自己定制的这些资源，也可以像骨架程序那样使用内置的风格。极小化光标风格由 API 函数 LoadIcone()装入，其原型如下：

HICON LoadIcon(HANDLE hInst,LPCSTR LpszName)；

这个函数返回一个图标的句柄。这里，hInst 为含有图标的模块的句柄，图标名字由 lps2-Name 指出。要使用内置图标，必须使第一个参数为 NULL，第二个参数用以下宏之一：

图标宏	形状
IDI_APPLICATION	缺省图标
IDI_ASTERISK	信息图标
IDI_EXLAMATION	叹号图标
IDI_HAND	停止图标
IDI_QUESTION	问号图标

使用 API 的 Load Cursor()函数装入图标，其原型如下：

此函数返回光标资源的句柄，在此，hInst 为包含鼠标的模块句柄，光标名字由 lpszName 指出。使用内置光标，必须使第一个参数为 NULL，第二个参数用以下宏之一，常用的内置光标有：

光标宏	形状
IDC_ARROW	缺省箭头
IDC_CROSS	十字光标
IDC_IBEAM	垂直 I 形光标
IDC_WAIT	沙漏光标。

骨架程序中创建的窗口背景色为亮灰色。刷子的句柄则由调用 API 函数 GetStockObject()获得。刷子是一种资源，使用预定的大小、颜色和图案。函数 GetStockObject()用于获取许多标准显示对象，包括刷子、画笔(用于画线)和字符字体等。其原型如下：

HGDIOBJ GetStockObject(int object)；

函数返回由 object 指出的对象的句柄，以下是程序可以选用的一些内置刷子：

宏名字	背景色
BLACK_BRUSH	黑色
DKGRAY_BRUSH	暗灰
HOLLOW_BRUSH	透明
LTGRAY_BRUSH	亮色
WHITE_BRUSH	白色

你可以使用这些作为 GetStockObject()的参数来获得刷子句柄。

一旦窗口类完全指定，就用 API 函数 RegisterClass()向 Windows NT 注册，其原型如下：

ATOM RegisterClass(LPWNDCLASS lpWClass)；

函数返回标识窗口类的值。ATOM 是用 typedef 定义的 int。每个窗口类都分配唯一的值。LPWNDCLASS 是指向 WNDCLASS 结构的指针。因此 lpWClass 必须是 WNDCLASS 结构的地址。

16 位转换注释　16 位 Windows 的 RegisterClass() 返回值是个布尔量，指出注册成功(true)或失败(false)。在 Windows NT 中它变为标识类的值。

2.6.2 创建窗口

一旦窗口类被定义并注册，应用程序就能用 API 函数 CreateWindow() 实际地创建一个那类的窗口。函数原型如下：

```
HWND CreateWindow (
  LPCSTR lpszClassName, /* name of window class */
  LPCSTR lpszWinName, /* title of window */
  DWORD dwStyle, /* type of window */
  int X,int Y, /* upper  left coordinates */
  int Width,int Height, /* dimensions of window */
  HWND hParent, /* handle of parent window */
  HMENU hMenu, /* handle of main menu */
  HINSTANCE hThisInst, /* handle of creator */
  LPVOID lpszAdditional /* pointer to additional info */
);
```

在骨架程序中，CreatWindow() 的许多参数是缺省的或是 NULL。实际上，通常 X，Y，Width 和 Height 使用宏 CW－USEDFAULT 就可以了，即告诉 Windows NT 为窗口选择合适的大小和位置。如果窗口没有父窗口，那么 hParent 必须是 NULL。(也可用宏 HWND－DESKTOP 作为参数。)如果窗口没有主菜单，那么 hMenu 必须是 NULL。如果没有额外的信息要求，那么 lpszAdditional 是 NULL。

剩下的四个参数必须由你的程序明确设定。首先，lpszClassName 必须指向窗口类的名字(就是在注册窗口类时用的名字)。lpszWinName 指向窗口标题，它可以是空串，但窗口一般都有标题。窗口的风格(或类型)由 dwstyle 的值决定。宏 WS_OVERLAPPEDWINDOW 是指具有系统菜单、边界、极大化和极小化方框的标准窗口，这一窗口风格最常见，但你也可以构造自己要求的风格，只需将不同风格的宏进行 OR 操作。以下是常见的一些风格：

风格宏	窗口特性
WS_OVERLAPPED	带有边界的重叠窗口
WS_MAXIMIZEBOX	极大化框
WS_MINIMIZEBOX	极小化框
WS_SYSMENU	系统菜单
WS_HSCROLL	水平滚动条
WS_VSCROLL	垂直滚动条

hThisInst 参数必须是应用程序当前实例的句柄。

CreateWindow() 返回其生成窗口的句柄，如果失败返回 NULL。