附录四　Windows 网络知识介绍

附 4.1　Windows Sockets

附 4.1.1　Windows Sockets 接口规范

Windows Sockets 规范以 U.C. Berkeley 大学 BSD UNIX 中流行的 Socket 接口为基准定义了一套 Windows 下网络编程接口。它不仅包含了人们所熟悉的 Berkeley Socket 风格的库函数，也包含了一组针对 Windows 的扩展库函数，以使程序员能充分地利用 Windows 消息驱动机制进行编程。

Windows Sockets 规范本意在于提供给应用程序开发者一套简单的 API，成为各家网络软件供应商共同遵循的标准。因此这份规范定义了应用程序开发者能够使用，并且网络软件供应商能够实现的一套库函数调用和相关语义。

遵守这套 Windows Sockets 规范的网络软件，我们称之为 Windows Sockets 兼容的，而 Windows Sockets 兼容实现的提供者，我们称之为 Windows Sockets 提供者。一个网络软件供应商必须百分之百地实现 Windows Sockets 规范,才能做到与 Windows Sockets 兼容。

任何能够与 Windows Sockets 兼容实现协同工作的应用程序就被认为是具有 Windows Sockets 接口。我们称这种应用程序为 Windows Sockets 应用程序。

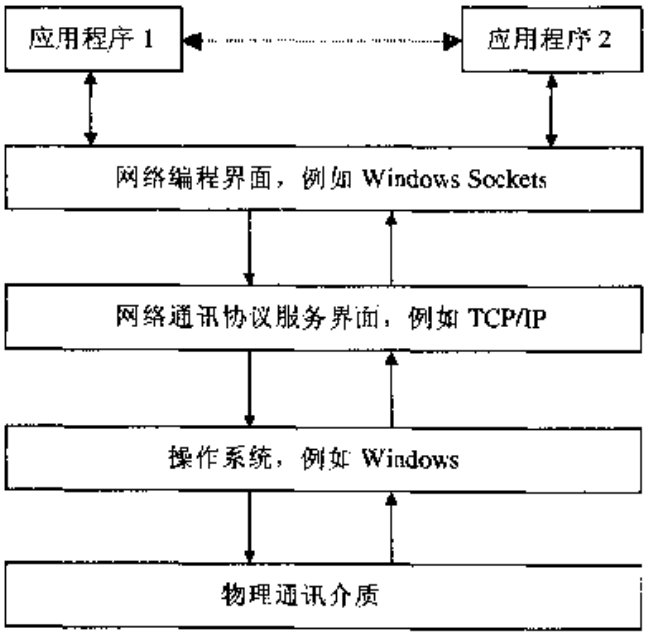
Windows Sockets 规范定义并记录了如何使用 API 与 Internet 协议族连接，尤其要指出的是所有的 Windows Sockets 实现都支持流套接口和数据报套接口。

应用程序调用 Windows Sockets 的 API 实现相互之间的通讯。Windows Sockets 又利用下层的网络通讯协议功能和操作系统调用实现实际的通讯工作。它们之间的关系如图附 4-1 所示。

虽然我们并不反对使用这一套 API 来实现另一通讯协议栈（而且期望在将来规范的修改中能够讨论这个问题），但这种用法已经超出了这一份规范所规定的范围，我们在此将不作讨论。

附 4.1.2　Bekeley 套接字

Windows Sockets 规范是建立在 Bekeley 套接字模型上。这个模型现在已是 TCP/IP 网络的标准。它给程序员提供了使用 UNIX 套接字进行网络编程的环境，并且简化了移植现有的基于套接字的应用程序源代码的工作，提高了网络程序的可移植性。Windows Sockets API 也是和 4.3BSD 的要求一致的。



图附 4-1 应用程序与 Windows Sockets 关系图

附 4.1.3 基于 Windows 平台的扩展

这一套 Windows Sockets API 能够在所有 3.0 以上版本的 Windows 和所有 Windows Scokets 实现上使用，所以它不仅为 Windwos Sockets 实现和 Windows Sockets 应用程序提供了 16 位操作环境，而且也提供了 32 位操作环境。

Windows Sockets 也支持多线程的 Windows 进程。一个进程包含了一个或多个同时执行的线程。在 Windows 3.1 非多线程版本中，一个任务对应了一个仅具有单个线程的进程。而我们在本书中所提到的线程均是指在多线程 Windows 环境中的真正意义的线程。在非多线程环境中（例如 Windows 3.0）,这个术语是指 Windows Sockets 进程。

Windows Sockets 规范中的针对 Windows 的扩展部分为应用程序开发者提供了开发具有 Windows 应用软件的功能。它有利于使程序员写出更加稳定并且更加高效的程序，也有助于在非占先 Windows 版本中使多个应用程序在多任务情况下更好地运作。除了 WSAStartup( )和 WSACleanup( )两个函数除外，其他的 Windows 扩展函数的使用不是强制性的。

附 4.1.4 这份规范的地位

Windows Sockets 是一份独立的规范。它的产生和存在是为了造益于应用程序开发者，网络软件供应商和广大计算机用户。这份规范的每一份正式出版的版本（非草稿）实际上代表了为网络软件供应商实现所需和应用程序开发者所用的

一整套 API。关于这套规范的讨论和改进还正在进行之中。这样的讨论主要是通过 Internet 上的一个电子邮件论坛——winsock@microdyne.com 进行的，同时也有不定期的会议举行。会议的具体内容会在电子邮件论坛上发表。

附 4.1.5　曾经作过的修改

1. Windows Sockets 1.0

Windows Sockets 1.0 代表了网络软件供应商和用户协会细致周到的工作的结晶。Windows Sockets 1.0 规范的发布是为了让网络软件供应商和应用程序开发者能够开始建立各自的符合 Windows Sockets 标准的实现和应用程序。

2. Windows Sockets 1.1

Windows Sockets 1.1 继承了 Windows Sockets 1.0 的准则和结构，并且仅在一些绝对必要的地方作了改动。这些改动都是基于不少公司在创作 Windows Sockets 1.0 实现时的经验和教训的。Windows Scokets 1.1 包含了一些更加清晰的说明和对 Windows Sockets 1.0 的小改动。此外 1.1 还包含了如下重大的变更：

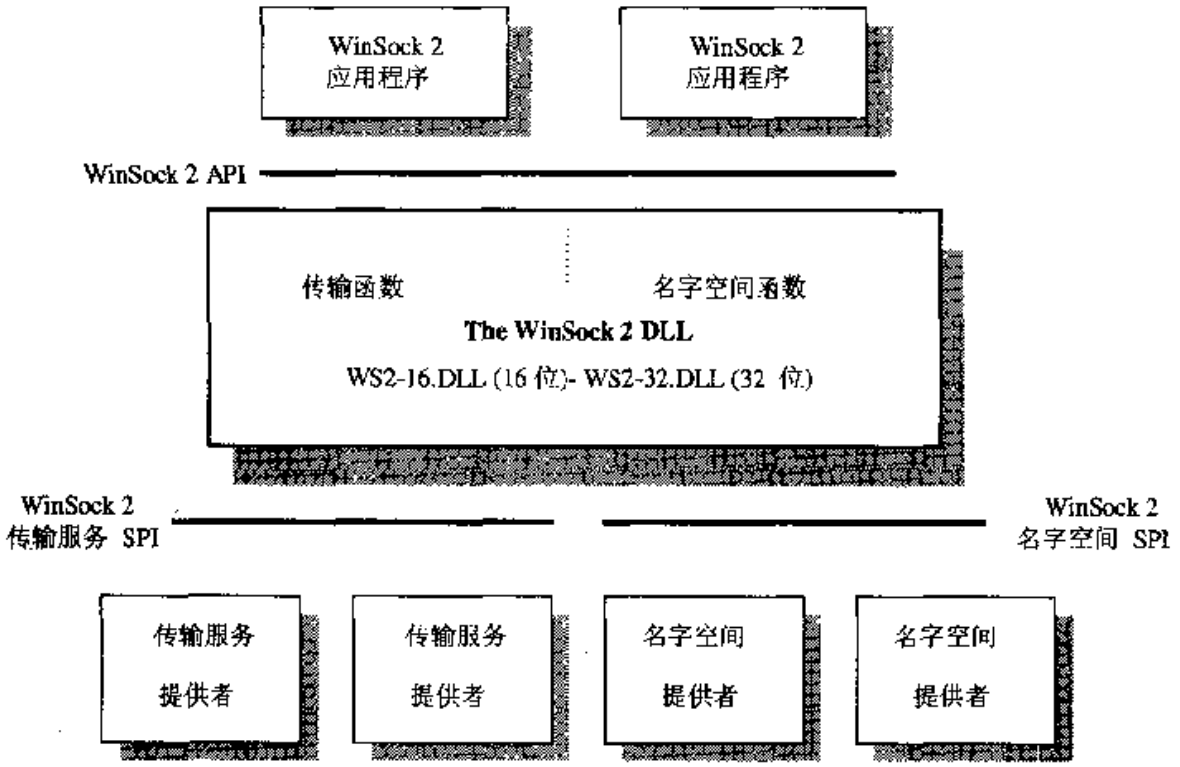
- 加入了 gethostname( )这个常规调用，以便更加简单地得到主机名字和地址。
- 定义 DLL 中小于 1000 的序数为 Windows Sockets 保留，而对大于 1000 的序数则没有限制。这使 Windows Sockets 供应商可以在 DLL 中加入自己的界面，而不用担心所选择的序数会和 Windows Scokets 将来的版本冲突。
- 增加了 WSAStartup( )函数和 WASClearup( )函数之间的关联，要求两个函数互相对应。这使得应用程序开发者和第三方 DLL 在使用 Windows Sockets 实现时不需要考虑其他程序对这套 API 的调用。
- 把函数 intr_addr( )的返回类型，从结构 in_addr 改为了无符号长整型。这个改变是为了适应 Microsoft C 编译器和 Borland C 编译器对返回类型为四字节结构的函数的不同处理方法。
- 把 WSAAsyncSelect( )函数语义从“边缘触发”改为“电平触发”。这种方式大大地简化了一个应用程序对这个函数的调用。
- 改变了 ioctlsocket( )函数中 FIONBIO 的语义。如果套接口还有未完成的 WSAAsyncSelect( )调用，该函数将失败返回。
- 为了符合 RFC 1122，在套接口选项中加入了 TCP_NODELAY 这一条。

3. Windows Socket 2.0

Windows Socket 2.0 就是现在 Windows 使用的版本。

（1）可以同时使用多个传输协议

为了用户能够同时使用多个传输协议，在 WindowsSocket 2 中，结构有所改变。在 Windows Sockets 1.1 中，软件开发商所提供的 DLL 实现了 Windows Sockets 的 API 和 TCP/IP 协议栈。Windows Sockets DLL 和底层协议栈的接口是惟一而且独占的。Windows Socket 2 改变了这种模型：它定义了一个 Windows Sockets DLL 和底层协议栈间的标准服务提供接口(SPI)。这使得一个 Windows Sockets DLL 能够同时访问不同软件开发商的多个底层协议栈。此外，Windows Sockets 2 并不像 Windows Sockets 1.1 仅支持 TCP/IP 协议栈。与 Windows 开放系统结构(WOSA)兼容的 Windows Sockets 2 的结构如图附 4-2 所示。



图附 4-2 Windows Socket 2 开放系统结构图

注意：16 位的 Windows Sockets 2 应用程序应使用 WS2-16.DLL，而 32 位的 Windows Sockets 2 应用程序应使用 WS2-32.DLL。但今后，为了简单起见，它们将都使用 WINSOCK.DLL。这并不会造成任何问题，因为在它们之间并没有任何语法上的区别。

由于以上的结构，现在已没有必要每个协议栈开发商都提供它们自己的 WINSOCK.DLL（甚至这样做也不是期望的）。因为任何一个 WINSOCK.DLL 能够在所有协议栈上工作。因此，WINSOCK.DLL 可以被看作是一个操作系统组件。Microsoft 将在 Windows 95 和 Windows NT 上提供一个 32 位的 WINSOCK.DLL。Intel 公司目前正在打算提供 Windows 3.1 和 Windows 3.11 上的

Windows Sockets 2 兼容的 16 位 WINSOCK.DLL。

（2）与 Windows Socket 1.1 应用程序的向后兼容性

Windows Socket 2 与 Windows Sockets 1.1 在两个基础上向后兼容：源码和二进制代码。这就实现了 Windows Sockets 应用程序和任何版本的 Windows Sockets 实现之间的最大的互操作性，而且也减少了 Windows Sockets 应用程序使用者，网络协议栈提供者和服务提供者的许多痛苦。现有的 Windows Sockets 1.1 兼容的应用程序可以在 Windows Sockets 2 实现上不加修改地运行，只要有一个 TCP/IP 协议栈被安装。

· 源码的兼容性

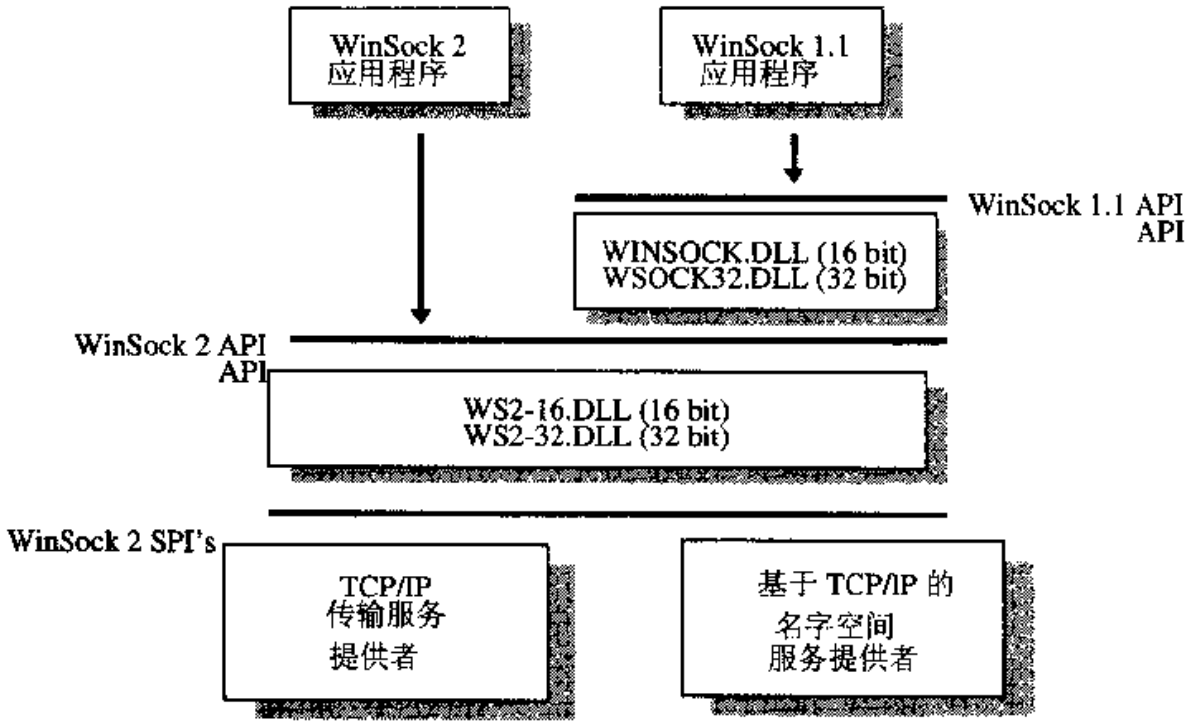
Windows Sockets 2 中的源码兼容性意味着所有的 Windows Sockets 1.1 版的 API 在 Windows Sockets 2 中都被保留了下来。这意味着现有的 Windows Sockets 1.1 应用程序的原程序可以被简单地移植到 Windows Sockets 2 系统上运行。程序员需要做的只是包含新的头文件——WINSOCK2.H 和简单地与合适的 Windows Sockets 2 函数库的连接。应用程序开发者应该把这种工作看作是完全转向 Windows Sockets 2 的第一步，因为有许多方式可以使用 Windows Sockets 2 中的新函数来提高原来的 Windows Sockets 1.1 应用程序的运行性能。

· 二进制兼容性

在设计 Windows Sockets 2 时的一个主要目标就是使得现有的 Windows Sockets 1.1 应用程序在二进制级别上能够不加修改地应用于 Windows Sockets 2 之上。由于 Windows Sockets 1.1 是基于 TCP/IP 上的，二进制兼容性就要求 Windows Sockets 2 系统提供基于 TCP/IP 上的 Windows Sockets 2 的传输和名字解析服务。为了 Windows Sockets 1.1 应用程序能在这种意义上运行，Windows Sockets 2 系统提供了一个附加的组件——Windows Sockets 1.1 的 DLL。Windows Sockets 2 安装时的提示保证了在终端机用户引进 Windows Sockets 2 系统时不会对已有的 Windows Sockets 软件环境有任何影响。

一个完全的 Windows Sockets 1.1 二进制兼容的必要的前提是在系统上已经安装了至少一个 TCP/IP 协议栈并且在 Windows Sockets 2 中进行了注册。Windows Sockets 1.1 目前通过 WSAData 结构中的某些元素来得到关于底层 TCP/IP 协议栈的信息（例如通过 WSAStartup()函数调用），这些信息包括 iMaxSockets，iMaxUdpDg 和 IPVendorInfo。但是在 Windows Sockets 2 中，应用程序应该知道忽略这些信息，因为这些值不能统一地适用于所有协议栈。不过 DLL 必须仍然提供这些值以免破坏 Windows Sockets 1.1 的应用程序。这些信息只能从（因此也只能应用于）缺省的 TCP/IP 服务提供者得到。缺省的 TCP/IP 服务提供者是由 WSAEnumProtocols( )调用返回的 PROTOCOL_INFO 结构缓冲区的第一条 TCP/IP

协议栈。



图附 4-3 与 Windows Sockets 1.1 二进制兼容性结构图

（3）在 Windows Sockets 中注册传输协议

要使 Windows Sockets 能够利用一个传输协议，该传输协议必须在系统上安装并且在 Windows Sockets 中注册。Windows Sockets 2 的 DLL 包含了一组 API 来完成这个注册过程。这个注册过程包括建立一个新的注册和取消一个已有的注册。在建立新的注册时，调用者（假设是协议栈开发商的安装程序）必须提供一组或多组完整的关于协议的信息，这些信息将被用来填充 PROTOCOL_INFO 结构。

• 使用多个协议

一个应用程序可以通过 WSAEnumProtocols( )功能调用来得到目前有多少个传输协议可以使用，并且得到与每个传输协议相关的信息，这些信息包含在 PROTOCOL_INFO 结构中。然而，某些传输协议可能表现出多种行为。例如 SPX 是基于消息的（发送者发送的消息的边界在网络上被保留了），但是接收的一方可以选择忽略这些边界并把套接口作为一个字节流来对待。这样就很合理地导致了 SPX 有两个不同的 PROTOCOL_INFO 结构条目，每一个条目对应了一种行为。

在 Windows Sockets 1 中仅有一个地址族（AF_INET），它包含了数量不多的一些众所周知的套接口类型和协议标识符。这在 Windows Sockets 2 中已经有所改变。除了现有的地址族，套接口类型和协议标识符原因兼容性被保留以外，

Windows Sockets 2 加入了许多惟一的但是可能并不为大家所知的地址族、套接口类型和协议标识符。不为大家所知并不意味着会对应用程序开发造成问题，因为一个企图做成协议无关的应用程序应该在对自身合适的基础上选择协议而不应该依赖于某个分配给它的特定的套接口类型或协议类型值。PROTOCOL_INFO结构中包含的通讯性质指明了协议的合适性（例如，基于消息的对应于基于字节流的，可靠的对应于不可靠的等）。基于合适性原则选取协议而不使用某个特定的协议名和套接口类型。

对于客户机/服务器模型，服务器一端的应用程序最好能够在所有合适的传输协议上建立监听套接口。这样，客户机一端的应用程序就可以通过任何合适的传输协议来与服务器一端的应用程序建立连接。这样做可以使得一个客户机应用程序易于移植。例如一台运行于 LAN 上的台式机的客户机应用程序在转到运行于无线网上的笔记本计算机时就不用作任何改变。

• select 函数应用中关于多个服务提供者的限制

在 Windows Sockets 2 中，函数 select( )使用 FD_SET 仅能应用于和单个服务提供者相连的套接口。但是这并不限制一个应用程序使用多个服务提供者打开多个套接口。如果应用程序开发者喜欢使用非阻塞方式编程，那么可以使用 WSAAsyncSelect( )函数。由于该函数需要一个套接口描述字作为输入参数，那么与该套接口相连的服务提供者是很重要的。如果一个应用程序需要在一组跨越多个服务提供者的套接口上使用带有阻塞语法的函数，那么应该使用 WSAWaitForMultiple Events( )函数。应用程序也可以使用 WSAEventSelect()函数。该函数允许应用程序把 FD_XXX 网络事件和一个事件对象相连接，并且在该事件对象中处理网络事件。

（4）协议无关的名字解析

Windows Sockets 2 包含了应用程序可以使用的多种标准化的网络名字服务。Windows Sockets 2 应用程序并不需要理解与名字服务相关的许多迥异的接口（例如 DNS，NIS，X.5000，SAP 等）。

（5）重叠 I/O 和事件对象

Windows Sockets 2 引入了重叠 I/O 的概念并且要求所有的传输协议提供者都支持这一功能。重叠 I/O 仅能在由 WSASocket( )函数打开的套接口上使用（使用 WSA_FLAG_OVERLAPPED 标记）。这种方式的使用将采用 Win32 建立的模型。

对于接收，应用程序使用 WSARecv( )函数或 WSARecvFrom( )函数来提供存放接收数据的缓冲区。如果数据在网络接收以前，应用程序已经提供了一个或多个数据缓冲区，那么接收的数据就可以立即被存放进用户缓冲区。这样可以省

去使用 recv( )函数和 recvfrom( )函数时需要进行的拷贝工作。如果在应用程序提供数据缓冲区时已经有数据到来，那么接收的数据将被立即拷贝进用户缓冲区。如果数据到来时，应用程序没有提供接收缓冲区，那么网络将回到我们熟悉的同步操作方式——传送来的数据将被存放进内部缓冲区，直到应用程序发出了接收调用并且提供了接收缓冲区，这时接收的数据就被拷贝进接收缓冲区。这种做法会有一个例外：就是当应用程序使用 setsockopt( )函数把接收缓冲区长度置为了 0 的时候。在这种情况下，对于可靠传输协议，只有在应用程序提供了接收数据缓冲区后，数据才会被接收；而对于不可靠传输协议，数据将会丢失。

对于发送的一方，应用程序使用 WSASend( )函数或 WSASendTo( )函数提供一个指向已填充的数据缓冲区的指针。应用程序不应在网络使用完该缓冲区的数据以前以任何方式破坏该缓冲区的数据。

重叠发送和接收调用会立即返回。如果返回值是 0，那么表明了 I/O 操作已经完成，对应的完成指示也已经可以得到。如果返回值是 SOCKET_ERROR，并且错误代码是 WSA_IO_PENDING，那么表明重叠操作已经被成功地初始化，今后发送缓冲区被用完或者接收缓冲区被填满时，将会有完成指示。任何其他的错误代码表明了初始化没有成功，今后也不会有什么完成指示。

发送操作和接收操作都可以被重叠使用。接收函数可以被多次调用，发出接收缓冲区，准备接收到来的数据。发送函数也可以被多次调用，组成一个发送缓冲区队列。要注意的是，应用程序可以通过按顺序提供发送缓冲区来确保一系列重叠发送操作的顺序，但是对应的完成指示有可能是按照另外的顺序排列的。同样，在接收数据的一方，缓冲区是按照被提供的顺序填充的，但是完成指示也可能按照另外的顺序排列。

WSAIoctl()函数（ioctolsocket()函数的增强版本）还可以使用重叠 I/O 操作的延迟完成特性。

- 事件对象

重叠 I/O 概念的引入需要建立一个机制使得应用程序能够正确地把发送和接收事件与今后它们完成时的指示相连接。在 Windows Sockets 2 中，这一点是通过事件对象实现的，它采用了 Win32 事件的模型。Windows Sockets 事件对象是一个相当简单的结构，它可以被创建、关闭、设置、清除、等待和检查。它们的主要用处是使得应用程序能够阻塞并等待直到一个或多个事件对象被设置。

应用程序可以使用 WSACreateEvent()函数来得到一个事件对象句柄，这个句柄可以作为以后的重叠发送和接收函数的输入参数（WSASend()，WSASendTo()，WSARecv()，WSARecvFrom()）。事件对象在创建时被清除，在相关的重叠 I/O 操作完成时由传输协议提供者设置（或者成功，或者出错）。每

个被 WSACreateEvent( )函数创建的事件对象都必须有对应的 WSACloseEvent( )函数释放它。

WSAEventSelect( )函数把一个或多个 FD_XXX 网络事件与一个事件对象连接。

在 32 位环境中，与事件对象相关的函数，包括 WSACreateEvent( )，WSACloseEvent( )，WSASetEvent( )，WSAResetEvent( )，WSAWaitForMultipleEvent( )和 WSAGetOverlappedResult( )，都被直接映射到对应的 Win32 函数，例如没有 WSA 前缀的同名函数。

· 接收操作完成指示

为了提供给应用程序适当的灵活性，Windows Sockets 2 为接收操作完成指示提供了多个选项。它们包括：等待（阻塞）事件对象，检查事件对象和套接字 I/O 完成例程。

· 阻塞并且等待完成指示

应用程序可以使用 WSAWaitForMultipleEvents( )函数来选择阻塞程序直到一个或多个事件对象被设置。在 Win16 实现中，这种方式将使用一个阻塞钩子，就像在标准的阻塞套接口操作时一样。在 Win32 实现中，进程或线程会被真正地阻塞。因为 Windows Sockets 2 事件对象被实现成 Win32 事件，所以 Win32 函数 WaitForMultipleObjects(章)也可以使用。这在线程需要阻塞套接口和非套接口事件时将会非常有用处。

· 检查完成指示

应用程序如果不希望使用阻塞方式，它可以使用 WSAGetOverlapped Results( )函数来检查与某个特定的事件对象相连的完成状态。该函数检查重叠操作是否完成，如果完成的话，处理重叠操作的出错信息，使得该信息在 WSAGetLastError( )函数调用时可以得到。

· 使用套接口 I/O 操作完成例程

所有的用来初始化重叠 I/O 操作的函数（WSASend()，WSASentTo()，WSARecv()，WSARecvFrom()）都把 lpCompletionRoutine 作为输入参数。这是一个应用程序定义的函数指针，在重叠 I/O 操作完成时可以被调用。

在 Win16 环境中，回调函数有可能在 VMM 环境（有时也被称作中断环境）下被激活。传送对时间要求较高的数据（例如视频或音频数据）在这种低延时、占先方式下接受这种指示会很方便。但是应用程序必须知道，在这种特殊情况下，只有很少一部分运行时和 Windows 库函数可以被调用。作为一条规则，应用程序必须把自己限制在一套 Windows 文档说明的在一个多媒体定时回调函数中可以被安全调用的运行时函数库。

在 Windows 95 和 Windows NT 中，完成例程与 Win32 文件 I/O 完成例程遵循着同样的规则。在所有的环境中，传输协议允许应用程序以完成例程的方式唤起发送和接收操作，而且保证对于给定的一个套接口，I/O 完成例程不会嵌套。这就允许了在一个占先的环境中进行对时间敏感的数据的传送。

- WSAOVERLAPPED 的细节

  WSAOVERLAPPED 结构提供了一个重叠 I/O 操作。WSAOVERLAPPED 结构被设计成与 Win32 中的 OVERLAPPED 结构兼容：

```
typedef struct WSAOVERLAPPED {
    DWORD     Internal;        // reserved
    DWORD     InternalHigh;    // reserved
    DWORD     Offset;          // ignored
    DWORD     OffsetHigh;      // ignored
    WSAEVENT  hEvent;
} WSAOVERLAPPED, LPWSAOVERLAPPED;
```

Internal 这一个保留的域是由重叠 I/O 实现的实体内部使用的。对于使用类文件方式创建套接口的传输服务提供者，这一域是被底层的操作系统使用的；对于其他的传输服务提供者（那些创建伪句柄的），可以根据需要使用这个域。

InternalHigh 这一个保留的域是由重叠 I/O 实现的实体内部使用的。对于使用类文件方式创建套接口的传输服务提供者，这一域是被底层的操作系统使用的；对于其他的传输服务提供者（那些创建伪句柄的），可以根据需要使用这个域。

Offset 由于套接口没有文件偏移量的概念，应用程序可以根据需要使用这个域。

· OffsetHigh 由于套接口没有文件偏移量的概念，应用程序可以根据需要使用这个域。

HEvent 如果一个重叠的 I/O 操作在被调用时没有使用 I/O 操作完成例程（lpCompletionRoutine 为空指针），那么这个域必须包含一个有效的 WSAEVENT 对象的句柄，否则（lpCompletionRoutine 不为空指针），应用程序可以视需要使用这个域。

（6）使用事件对象异步通知

为了适应一些应用程序例如向导程序或者某些没有用户界面的服务程序（因此不使用窗口句柄），Windows Socket 2 提供了 WSAEventSelect( )函数和