

第二部分 Winsock API

本书第二部分讲述的是在 Win32平台上的 Winsock编程。对于众多的基层网络协议，Winsock是访问它们的首选接口。而且在每个 Win32平台上，Winsock都以不同的形式存在着。Winsock是网络编程接口，而不是协议。它从 Unix平台的Berkeley（BSD）套接字方案借鉴了许多东西，后者能访问多种网络协议。在 Win32环境中，Winsock接口最终成为一个真正的“与协议无关”接口，尤其是在 Winsock 2发布之后。

接下来的三章讲解了协议和 Winsock的基本知识，其中包括每种协议的定址方案，以及一个简单的 Winsock客户机 / 服务器示例。再后面的几章讲述的是 Winsock 2中的一些新特性，比如传送服务提供者、命名空间提供者和服务质量（QoS）等等。就其中的一部分技术来说，可能会存在一些容易混淆的地方。原因是尽管它们都在 Winsock 2规范中得到了定义，而 Winsock 2在目前所有的Win32平台上都已得到支持（Windows CE除外，本部分还会对其详述），但落实到具体的平台，却并非所有这些特性都已真正地实现。必要时，我们会指出具体的限制在哪里。本部分开始之前，我们假定大家已具备了 Winsock（或BSD套接字）的基本知识，而且多少熟悉一些客户机 / 服务器的 Winsock基本知识。

第5章 网络原理和协议

建立Winsock 2规范的主要目的是提供一个与协议无关的传送接口。在各种不同的网络传送协议上，假若能为网络编程提供一个大家都很熟悉的接口，当然是一件不错的事情。但尽管如此，仍需注意各种网络协议的一些特征。本章将全面讲述使用特定协议时应该留意的一些特征，其中包括一些基本的网络连接原理。另外，我们还将讨论如何通过程序向 Winsock查询协议信息，并探讨针对一种具体协议创建套接字所需的基本步骤。

5.1 协议的特征

本章第一小节着重讲解目前常用网络传送协议的一些基本特征。通过这里的学习，大家可掌握与协议行为类型有关的一些背景知识。同时，作为程序员，可大致知道特定协议在程序中的行为方式。

5.1.1 面向消息

对每个离散写命令来说，如果传送协议把它们（而且只有它们）当做一条独立的消息在网上传送，我们就说该协议是面向协议的。同时，还意味着接收端在接收数据时，返回的数据是发送端写入的一条离散消息。接收端不能得到更多的消息，仅此而已。比如，在图 5-1中，左边的工作站向右边的工作站提交了三条分别是 128、64和32字节的消息。作为接收端的工作站发出三条读取命令，缓冲区是 256个字节。后来的各次调用返回的分别是 128、64和32个字

节。第一次读取调用不会将这所有的三个数据包都返回，即使这些数据包已经收到也如此。这称为“保护消息边界”(preserving message boundaries)，一般出现在交换结构化数据时。网络游戏是“保护消息边界”的较好范例。每个玩家均向别的玩家发出一个带有地图信息的数据包。这种通信后面的代码很简单：一个玩家请求一个数据包，另一个玩家又准确地从别的玩家处获得一个地图信息数据包。

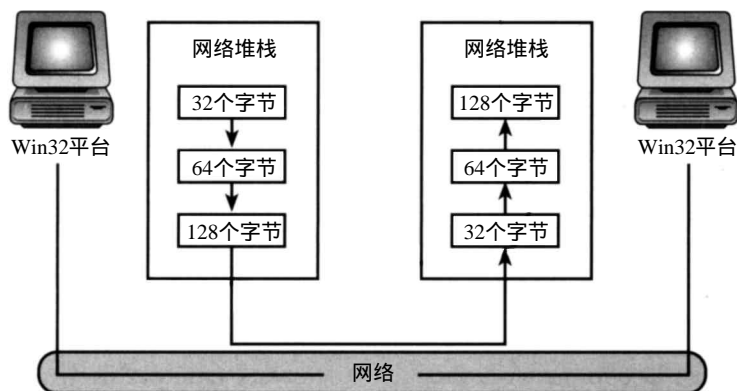


图5-1 数据报服务

无保护消息边界的协议通常称作“基于流的协议”。大家要知道“基于流的协议”这一术语常用来指代附加特性。流服务的定义是连续的数据传输；不管消息边界是否存在，接收端都会尽量地读取有效数据。对发送端来说，意味着允许系统将原始消息分解成小消息或把几条消息积累在一起，形成一个较大的数据包。对接收端来说，则是数据一到达网络堆栈，网络堆栈就开始读取它，并将它缓存下来等候进程处理。在进程请求处理大量数据时，系统会在不溢出为客户请求提供的缓冲区这一前提下，尽量返回更多的数据。在图 5-2中，发送端提交了三个数据包：分别是 128、64和32个字节；但是，本地系统堆栈自由地把这些数据聚合在一起，形成一个大的数据包。这种情况下，重组后的 2 个数据包就会一起传输。是否将各个独立的数据包累积在一起受许多因素的影响，比如最大传输单元或 Nagle算法。在TCP/IP中，在将积累起来的数据发到线上之前，Nagle算法在等候数据积累的主机中进行。在需要发送的数据积累到一定数量或预定时间已超过之前，这个主机会一直等下去。实施Nagle算法时，主机的通信方在发送主机确认之前，会等一等外出数据，主机的通信方就不必发送一个只有确认的数据包。发送小数据包不仅没有多少意义，而且还会徒增错误检查和确认，相当烦人。

在接收端，网络堆栈把所有进来的数据包聚集在一起，归入既定进程。我们来看看图 5-2。如果接收端执行一次 256字节缓冲区的读取，系统马上就会返回 224个字节。如果接收端只要求读取20个字节，系统就会只返回 20个字节。

伪流

伪流 (pseudo-stream) 这个术语常用于某种系统中，该系统使用的协议是基于消息的，发送的数据分别在各自独立的数据包内，接收端读取并把消息缓存在一起，这样，接收应用程序便可读取任意大小的数据块。把图 5-1中的发送端和图5-2中的接收端结合起来，便可说明伪流的工作原理。发送端必须分别发送各自独立的数据包，但接收端可以对收到的数据包自由组合。一般情况下，可把伪流视作一个普通的“面向流的协议”。

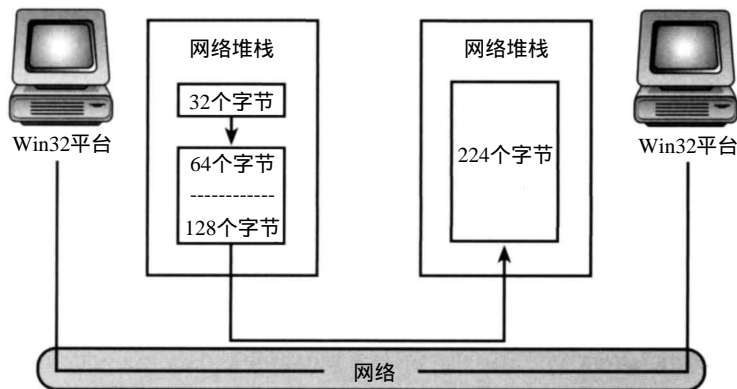


图5-2 流服务

5.1.2 面向连接和无连接

通常情况下，一个协议提供面向连接的服务，或提供无连接的服务。面向连接的服务中，进行数据交换之前，必须与通信方建立一条路径。这样既确定了通信方之间存在路由，又保证了通信双方都是活动的、都可彼此响应，但其特点是在通信双方之间建立一个通信信道需要很多开支。除此以外，大部分面向连接的协议为保证投递无误，可能会因为执行额外的计算来验证正确性，因此，进一步增加开支。而无连接协议却不保证接收端是否正在收听。无连接服务类似于邮政服务：发信人把信装入邮箱即可。至于收信人是否想收到这封信或邮局是否会因为暴风雨未能按时将信件投递到收信人处等等，发信人都不得而知。

5.1.3 可靠性和次序性

在设计用于特定协议的应用程序来说，可靠性和次序性是我们必须了解的最具决定性的特性。大多数情况下，可靠性和次序性与协议是无连接的，还是面向连接的密切相关。可靠性（或保证投递）保证了发送端发出的每个字节都能到达既定的接收端。不具备可靠性的协议则不能保证每个字节都能到达接收端，同样不能保证数据的完整性。

次序性是指对数据到达接收端的顺序进行处理。保护次序性的协议保证接收端收到数据的顺序就是数据的发送顺序。显然，没有保护次序性的协议就没有次序保证。

在面向连接的通信中，如果你一直试图在通信双方建立一个显式通信信道，一般希望协议能够保证数据的完整性和次序性。多数情况下，面向连接的协议的确能保证数据的可靠性。下一小节，我们将讨论真正的协议及其特征。注意，可靠性和次序性两者不能兼而得之，保证了数据包顺序，就不能自动保证数据的完整性。当然，无连接协议的过人之处就是速度：对接入接收端的虚拟连接来说，它们不会影响其建立。为什么这样反而降低了错误检查的速度呢？原来这便无连接协议一般不保证数据完整性和次序性，而面向连接的协议却能做到的原因。数据报如此不完善，为什么大家都喜欢用它呢？原来，无连接协议按数量大小排序，比面向连接的通信快得多。另外，它不必去验证数据完整性和数据收到确认与否和发送少量数据时增加的复杂性等等。对非临界的数据传输来说，数据报非常有用。而前面讨论过的网络游戏示例之类的应用，数据报简直是为它们量身定做的：每个玩家都可利用数据报周期性地向别的玩家发送他/她在游戏中的位置。如果某一个客户机丢失了一个数据包，很快就能

收到另外一个，像无缝通信一样方便。

5.1.4 从容关闭

从容关闭只出现在面向连接的协议中。在这种关闭过程中，一方开始关闭通信会话，但另一方仍然可以读取线上或网络堆栈上已挂起的数据。如果面向连接的协议不支持从容关闭，只要其中一方关闭了通信信道，都会导致连接立即中断，数据丢失，接收端不能读取数据这些情况出现。如果使用 TCP 协议，连接双方都必须执行一次关闭，以便完全中断连接。发起方向另一个通信方发出一个带有 FIN 控制标志的分段（数据报）。接收端的通信方则向发起方返回一个 ACK 控制标志，确认收到 FIN，但此时发起方仍然可发送数据。FIN 控制标志表示发起关闭的这一方不再发送数据。一旦通信方决定不需要发送数据，它就会发出一个 FIN 控制标志，并带有发起方确认的 ACK 控制标志。这时，该连接已完全关闭。

5.1.5 广播数据

广播数据即数据从一个工作站发出，局域网内的其他所有工作站都能收到它。这一特征适用于无连接协议，因为 LAN 上的所有机器都可获得并处理广播消息。使用广播消息的不利之处是每台机器都必须对该消息进行处理。比如，一用户在 LAN 上广播一条消息，每台机器上的网卡都会收到这条消息，并把它上传到网络堆栈。然后，堆栈将这条消息在所有的网络应用中循环，看它们是否应该接收这条消息。通常，这个局域网上的多数机器对该消息都不感兴趣，草草地一弃了之。但是，各台机器仍需花时间来处理这个数据包，看是否有应用对它感兴趣。结果，高广播通信流使 LAN 上的机器陷入困境，因为每个工作站都要检查这个数据包。一般情况下，路由器都不会传送广播包。

5.1.6 多播数据

多播是指一个进程发送数据的能力，这些数据即将由一个或多个接收端进行接收。进程加入一个多播会话的方法和采用的基层协议有关。比如，IP 协议下，多播是广播的一种变形。IP 多播要求对收发数据感兴趣的所有主机加入一个特定的组。进程希望加入多播组时，网卡上会增添一个过滤器，这样，只有绑定组地址的数据才会被网络硬件捡起，并上传到网络堆栈进行恰当处理。视频会议应用常常使用多播。第 11 章将详细论述 Winsock 的多播编程和其他一些关键问题。

5.1.7 服务质量

服务质量（QoS）是应用的一种能力，用以请求针对专门用途分配特定的带宽。服务质量的好处可从实时视频流式传输中一见端倪。对实时视频流式传输应用的接收端而言，要显示平滑清晰的图像，发送的数据必须限定在特定范围内。过去，应用是将数据缓存下来，再从缓冲区中重播帧，进而获得流畅的视频。如果收到数据的时间太久，重播例程就可拥有大量可播放的、缓存起来的帧。QoS 允许网络上预留的带宽，因此，可以在保证的强制范围内，离线读取帧。理论上讲，这意味着同一个实时视频流式传输应用可用 QoS 消除缓存帧的必要。我们将在第 12 章详细讨论 QoS。

5.1.8 部分消息

部分消息只用于面向消息的协议。比如说，一个应用想接收消息，但本地计算机已收到了它的部分数据。这是相当常见的，特别是发送端计算机正在传输大型消息时。本地计算机可能没有足够多的资源来容纳整条消息。实际上，多数面向消息的协议对数据报的最大长度都有一个合理限制，因此，这一特殊事件不会经常发生。但是，大多数数据报协议都支持大型消息——大的足以需要根据物理方法来将其分解成许多小传输块。如此一来，用户的应用请求读取这条消息时，就有这种可能性存在——用户系统可能只收到了部分消息。如果特定协议支持部分消息，就会通知读取方“已返回的数据只是整条消息中的一部分”。如果协议不支持部分消息，基层网络堆栈就会坚持接收其余的消息，直到收完为止。如果由于某种原因，整条消息不能全部到达，那么收到的数据报就会不完整，多数不支持部分消息的不可靠协议都会把这个不完整的数据报简单地丢弃。

5.1.9 路由选择的考虑

一个重要考虑就是协议是否可路由。如果协议可路由，就可在两个工作站之间建立一条成功的通信路径（要么是面向连接的回路，要么是数据报的数据路径），不管这两个工作站之间存在的网络硬件是什么。比如，机器 A 和机器 B 分别在各自的子网中。连接两个子网的路由器 A 把这两台机器分开了。一个可路由的协议意识到机器 B 和机器 A 不在同一个子网上：它就会把数据包定向到路由器，由路由器来决定数据的最佳转发方式，以便数据能抵达机器 B。非路由协议不能作出这样的规定：路由器会将它收到的非路由协议数据包统统丢弃。路由器不对发自非路由协议的数据包进行转发，即便数据包的既定目的地在其连接的子网上。

NetBEUI 是 Win32 平台支持的唯一的一个协议，它不能路由。

5.1.10 其他特征

Win32 平台上支持的各个协议都展现了特定的或独特的特征。同时，还有大量不胜枚举的其他特性，比如字节序和最大传输单元，都可用来说明目前网络上所用的各种协议。然而，对编写 Winsock 应用程序来说，并不一定要面面俱到，囊括所有特征。Winsock 2 提供了一种功能，可把各个实用的协议提供者及其特征列举出来。本章 5.3 节对这一功能进行解释，并展示了一个代码实例。

5.2 支持的协议

Win32 平台提供的最有用的特征之一是能够同步支持多种不同的网络协议。正如大家在第 2 章中看到的那样，Windows 重定向器保证将网络请求路由到恰当的协议和子系统；但是，有了 Winsock，就可以编写可直接使用任何一种协议的网络应用程序了。第 6 章将讨论利用适用于某一工作站的种种协议，以及如何为网络中的机器定址的问题。利用 Winsock 编程接口的好处之一是因为它是一个与协议无关的接口。不管使用的是哪一种协议，它们的操作大多数是相通的。尽管如此，要为网络通信定位和连接通信方，仍然必须了解如何为工作站定址。

5.2.1 支持的 Win32 网络协议

Win32 平台支持相当多的协议。通常，各个协议都同时具备多种行为。比如，网际协议

(IP) 既支持面向连接的流服务, 又支持无连接的数据报服务。表 5-1 提供了大部分实用协议及各协议支持的一些行为。

表5-1 实用协议的特性

协 议	名 称	消息 类型	连接 类型	可靠性	数据包的 次序性	从容 关闭	广播 支持	多点传 输支持	服务 质量	最大消息 量(字节)
IP	MSAFD TCP	流	连接	有	有	有	无	无	无	无
	MSAFD UDP	消息	无连接	无	无	无	有	有	无	65467
	RSVP TCP	流	连接	有	有	有	无	无	有	无
	RSVP UDP	消息	无连接	无	无	无	有	有	有	65467
IPX/SPX	MSAFD	消息	无连接	无	无	无	有	有	无	576
	nwlkpx[IPX]									
	MSAFD	消息	连接	有	有	无	无	无	无	无
	nwlksp[SPX]									
	MSAFD	消息	连接	有	有	无	无	无	无	无
	nwlksp[SPX][伪流]									
	MSAFD	消息	连接	有	有	有	无	无	无	无
	nwlksp[SPXII]									
NetBIOS	MSAFD	消息	连接	有	有	有	无	无	无	无
	nwlksp[SPXII][伪流]									
	连续数据包	消息	连接	有	有	无	无	无	无	64KB (65535)
AppleTalk	数据报	消息	无连接	无	无	无	有 (SP25)	无	无	64KB (65535)
	MSAFD	消息	连接	有	有	有	无	无	无	64KB (65535)
	AppleTalk[ADSP]									
	MSAFD	消息	连接	有	有	有	无	无	无	无
	AppleTalk[ADSP][伪流]									
	MSAFD	消息	连接	有	有	有	无	无	无	4096
	AppleTalk[PAP]									
	MSAFD	流	无连接	无	无	无	无	无	无	无
ATM	AppleTalk[RTMP]									
	MSAFD	流	无连接	无	无	无	无	无	无	无
	AppleTalk[ZIP]									
红外线 套接字	MSAFD ATM AAL5	流	连接	无	有	无	无	有	有	无
	本地 ATM(AAL5)	消息	连接	无	有	无	无	有	有	无
	MSAFD Irda[IrDA]	流	连接	有	有	有	无	无	无	无

NetBIOS支持发送到一个或一组NetBIOS客户机的数据报。但不支持全面广播。

5.2.2 Windows CE网络协议

Windows CE和其他 Win32平台不同, 它只支持 TCP/IP网络传输协议。除此以外, Windows CE只支持 Winsock 1.1规格, 这意味着本小节讨论的 Winsock 2特性大多不能用于这个平台。Windows CE通过一个重定向器支持依赖于 TCP/IP协议进行通信的 NetBIOS, 但不会通过本地 NetBIOS API或 Winsock 为 NetBIOS 提供编程接口。

5.3 Winsock 2协议信息

针对指定的工作站上安装哪种协议和各种协议特性的返回问题, Winsock 2提供了一种解

决办法。如果一个协议支持多种行为，每类行为在系统中都有各自的目录条目。比如，如果在自己的系统中安装了 TCP/IP，系统中就会有二个 IP 条目：一个条目针对 TCP，是可靠的而面向连接的；而另一个条目针对 IP，是不可靠且有无连接的。

要想获得系统中安装的网络协议的相关信息，调用这个函数 WSAEnumProtocols 即可，并像这样定义它：

```
int WSAEnumProtocols (
    LPINT lpiProtocols,
    LPWSAPROTOCOL_INFO lpProtocolBuffer,
    LPDWORD lpdwBufferLength
);
```

这个函数取代了 Winsock 1.1 中的 EnumProtocols 函数，EnumProtocols 函数是 Windows CE 中必不可少的函数。唯一的区别是：WSAEnumProtocols 返回的是一个 WSAPROTOCOLS_INFO 结构的数组；而 EnumProtocols 返回的则是 PROTOCOL_INFO 结构的数组，该数组中包含的字段少于 WSAPROTOCOLS_INFO 结构中的字段（但信息是差不多的）。WSAPROTOCOL_INFO 结构的格式如下：

```
typedef struct _WSAPROTOCOL_INFOW {
    DWORD          dwServiceFlags1;
    DWORD          dwServiceFlags2;
    DWORD          dwServiceFlags3;
    DWORD          dwServiceFlags4;
    DWORD          dwProviderFlags;
    GUID           ProviderId;
    DWORD          dwCatalogEntryId;
    WSAPROTOCOLCHAIN ProtocolChain;
    int            iVersion;
    int            iAddressFamily;
    int            iMaxSockAddr;
    int            iMinSockAddr;
    int            iSocketType;
    int            iProtocol;
    int            iProtocolMaxOffset;
    int            iNetworkByteOrder;
    int            iSecurityScheme;
    DWORD          dwMessageSize;
    DWORD          dwProviderReserved;
    WCHAR          szProtocol[WSAPROTOCOL_LEN + 1];
} WSAPROTOCOL_INFOW, FAR * LPWSAPROTOCOL_INFOW;
```

打开 Winsock

在可以调用一个 Winsock 函数之前，必须先加载一个版本正确的 Winsock 库。Winsock 启动例程是 WSAStartup，它的定义是：

```
int WSAStartup(WORD wVersionRequested, LPWSADATA lpWSADATA)
```

第一个参数是准备加载的 Winsock 库的版本号。就目前的 Win32 平台而言，Winsock 2 库的最新版本是 2.2。唯一的例外是 Windows CE，它只支持 Winsock 1.1 版。如果需要 Winsock 2.2 版，指定这个值（0x0202）或使用宏 MAKEWORD(2,2) 即可。高位字节指定副版本，而低位字节则指定主版本。

第二个参数是 WSADATA 结构，它是调用完成之后立即返回的。WSADATA 包含了

WSAStartup加载的关于 Winsock 版本的信息。表 5-2 列出了 WSADATA 结构的各个字段，WSADATA 结构的真正定义是：

```
typedef struct WSADATA {  
    WORD        wVersion;  
    WORD        wHighVersion;  
    char        szDescription[WSADESCRIPTION_LEN + 1];  
    char        szSystemStatus[WSASYS_STATUS_LEN + 1];  
    unsigned short iMaxSockets;  
    unsigned short iMaxUdpDg;  
    char FAR *    lpVendorInfo;  
} WSADATA, FAR * LPWSADATA;
```

大致说来，在 WSADATA 结构中，返回的唯一有用的信息是 wVersion 和 wHighVersion。属于最大套接字和最大 UDP 长度的条目应该从自己正在使用的特定协议目录条目中获取。上一小节曾谈到这一点。

表5-2 WSADATA结构的成员字段

字 段	说 明
wVersion	调用者希望使用的 Winsock 版本号
wHighVersion	加载的 Winsock 库所支持的最高 Winsock 版本，通常和 wVersion 的值相同
szDescription	加载的 Winsock 库的文本说明
szSystemStatus	一个文字串，其中包含相应的状态或配置信息
iMaxSockets	套接字的最大编号（Winsock 2 或稍后的版本忽略了该字段）
iMaxUdpDg	UDP 数据报的最大容量（Winsock 2 或稍后的版本忽略了该字段）
lpVendorInfo	厂商专有信息（Winsock 2 或稍后的版本忽略了该字段）

在结束 Winsock 库，而且不再需要调用任何 Winsock 函数时，附带例程会卸载这个库，并释放资源。这个函数的定义是：

```
int WSACleanup (void);
```

记住，每次调用 WSAStartup，都需要调用相应的 WSACleanup，因为每次启动调用都会增加对加载 Winsock DLL 的引用次数，它要求调用同样多次的 WSACleanup，以此抵消引用次数。

注意，Winsock 2 与 Winsock 1.1 规格中的所有函数调用完全兼容。因此，对 Winsock 1.1 规格编写的应用程序来说，在加载 Winsock 2 库的情况下，该应用程序也能够运行，因为 Winsock 1.1 函数通过 Winsock 2 对应函数得以映射。

要调用 WSAEnumProtocols 函数，最简单的方法是利用相当于 NULL 和 lpdwBufferLength 的 lpProtocolBuffer，令初次调用为 0。调用失败，返回 WSAENOBUFS 错误，但 lpdwBufferLength 中包含了恰如其分的缓冲区长度（这一长度是返回所有协议信息所需的）。一旦分配了恰当的缓冲区长度，便可利用这个缓冲区进行下一次调用，该函数返回 WSAPROTOCOL_INFO 结构所返回的号码。这时，就可逐步深入各结构，查找自己所需要的协议条目了。本书配套 CD-ROM 上的实例程序 Enum.c 列举了所有已安装的协议及其特性。

WSAPROTOCOL_INFO 结构最常用的字段是 dwServiceFlags1，它是代表不同协议属性的一个位字段。表 5-3 列出了可在字段中设置的各种位标志，并对各属性的含义进行了说明。要查看特定属性，先选定相应的属性标志，然后对该属性和 dwServiceFlags1 字段进行按位和运

算。如果结果值非零，这个属性就会出现在指定协议中；反之，则不会出现。

表5-3 协议标志

属 性	含 义
XPI_CONNECTIONLESS	该协议提供无连接的服务。如果不设，则支持面向连接的数据传输
XPI_GUARANTEED_DELIVERY	该协议保证发送出去的所有数据都将到达既定接收端
XPI_GUARANTEED_ORDER	保证数据按其发送顺序到达接收端，且数据不会重复。但是，该协议不能保证投递的准确性
XPI_MESSAGE_ORIENTED	实现消息边界
XPI_PSEUDO_STREAM	该协议是面向消息的，但接收端会忽略消息边界
XPI_GRACEFUL_CLOSE	支持二相关闭：通知各个通信方另一方打算关闭通信信道。如果不设，则只执行失败关闭
XPI_EXPEDITED_DATA	该协议提供紧急数据（带外数据）
XPI_CONNECT_DATA	该协议支持带有连接请求的数据传输
XPI_DISCONNECT_DATA	该协议支持带有无连接请求的数据传输
XPI_SUPPORT_BROADCAST	该协议支持广播机制
XPI_SUPPORT_MULTIPOINT	该协议支持多点或多播机制
XPI_MULTIPOINT_CONTROL_PLANE	如果设置了这个协议标志，就会启动控制面板。反之，则不启动
XPI_MULTIPOINT_DATA_PLANE	如果设置了这个协议标志，就会启动数据面板。反之，则不启动
XPI_QOS_SUPPORTED	该协议标志支持QoS请求
XPI_UNI_SEND	该协议在发送方向上是单向的
XPI_UNI_RECV	该协议在接收方向上是单向的
XPI_IFS_HANDLES	提供者返回的套接字描述符是Installable File System（可安装文件系统）句柄，可用于ReadFile WriteFile之类的API函数中
XPI_PARTIAL_MESSAGE	WSASend和WSASendTo中均支持MSG_PARTIAL标志

大多数协议标志，都将在下面的一章或几章中讨论，所以我们在此不详细介绍各标志的全部含义。其他重要字段是iProtocol、iSocketType和iAddressFamily。iProtocol字段定义该条目属于哪个协议。对支持多种行为的协议来说（比如说面向流的连接或数据报连接）iSocketType字段非常之重要。最后，iAddressFamily字段用于为指定协议区分正确的定址结构。在为指定协议建立套接字时，这三个条目都很重要，我们将在下一小节重点介绍。

5.4 Windows套接字

现在，大家对各种协议及其属性都比较熟悉了。接下来看看这些协议在Winsock中的用法。如果熟悉Winsock，就知道API是建立在套接字基础上的。所谓套接字，就是一个指向传输提供者的句柄。Win32中，套接字不同于文件描述符，所以它是一个独立的类型——SOCKET。套接字是由两个函数建立的：

```
SOCKET WSASocket (
    int af,
    int type,
    int protocol,
    LPWSAProtocolInfo lpProtocolInfo,
    GROUP g,
    DWORD dwFlags
);

SOCKET socket (
```

```

int af,
int type,
int protocol
);

```

第一个参数 *af*，是协议的地址家族。比如，如果想建立一个 UDP 或 TCP 套接字，可用常量 *AF_INET* 来指代互联网协议（IP）。第二个参数 *type*，是协议的套接字类型。套接字的类型可以是下面五个值：*SOCK_STREAM*、*SOCK_DGRAM*、*SOCK_SEQPACKET*、*SOCK_RAW* 和 *SOCK_RDM*。第三个参数是 *protocol*。指定的地址家族和套接字类型有多个条目时，就可用这个字段来限定使用特定传输。表 5-4 列出的值用于针对一个指定网络传输的地址家族、套接字类型和协议字段。

表5-4 套接字参数

协 议	地址家族	套接字类型	协议字段
Internet Protocol (IP)	<i>AF_INET</i>	TCP	<i>IPPROTO_IP</i>
		UDP	<i>IPPROTO_UDP</i>
		Raw sockets	<i>IPPROTO_RAW</i>
IPX/SPX	<i>AF_NS</i>	MSAFD	<i>NSPROTO_IPX</i>
		nwlknipx[IPX]	
	<i>AF_IPX</i>	MSAFD	<i>NSPROTO_SPX</i>
		nwlknsp[SPX]	
		MSAFD	<i>NSPROTO_SPX</i>
		nwlknsp[SPX]	
		[Pseudo-stream]	
		MSAFD	<i>NSPROTO_SPXII</i>
		nwlknsp[SPXII]	
		MSAFD	<i>NSPROTO_SPXII</i>
		nwlknsp[SPXII]	
		[Pseudo-stream]	
NetBIOS	<i>AF_NETBIOS</i>	Sequential Packets	<i>LANA number</i>
AppleTalk	<i>AF_APPLETALK</i>	Datagrams	<i>LANA number</i>
		MSAFD	<i>ATPROTO_ADSP</i>
		AppleTalk[ADSP]	
		MSAFD	<i>ATPROTO_ADSP</i>
		AppleTalk[ADSP]	
		[Pseudo-stream]	
		MSAFD	<i>ATPROTO_PAP</i>
		AppleTalk[PAP]	
		MSAFD	<i>DDPPROTO_RTMP</i>
		AppleTalk[RTMP]	
ATM	<i>AF_ATM</i>	MSAFD	<i>DDPPROTO_ZIP</i>
		AppleTalk[ZIP]	
		MSAFD ATM AAL5	<i>ATMPROTO_AAL5</i>
		Native ATM(AAL5)	<i>ATMPROTO_AAL5</i>
Infrared Sockets	<i>AF_IRDA</i>	MSAFD Irda[IrDA]	<i>IRDA_PROTO</i> <i>_SOCK_STREAM</i>

建立套接字的前三个参数组织成三级。第一个同时也是最重要的参数是地址家族。它指定准备使用哪种协议，另外还为第二和第三个参数指定有效选项。比如，如果选择了 ATM 地址家族 (AF_ATM)，那么在选用套接字类型时，就会限定只能采用原始套接字 (SOCK_RAW)。同样，在选定一个地址家族和套接字类型后，也会限定只能用哪种协议。但是，投递协议参数为 0 是可行的。然后，系统再根据其他两个参数——af 和 type，选定一个传输提供者。列举协议条目时，要检查 WSAPROTOCOL_INFO 结构的 dwProviderFlags 条目，如果 socket 或 WSASocket 协议参数是 0，它就是即将使用的默认传输。

如果在 WSASocket 函数时，已经利用 WSAEnumProtocols 列举了所有协议，就可选定一个 WSAPROTOCOL_INFO 结构，并将它当作 lpProtocolInfo 参数投递到 WSASocket。之后，若在前三个参数 (af、type 和 protocol) 中都指定常量 FROM_PROTOCOL_INFO，就会转而采用 WSAPROTOCOL_INFO 结构中提供的那三个值。以上便是教大家如何指定一个准确无误的协议条目。

最后两个 WSASocket 标志很简单。组参数始终为 0，因为目前尚无支持套接字组的 Winsock 版本。要指定一个或多个下列标志，可用 dwFlags 参数：

```
WSA_FLAG_OVERLAPPED
WSA_FLAG_MULTIPOINT_C_ROOT
WSA_FLAG_MULTIPOINT_C_LEAF
WSA_FLAG_MULTIPOINT_D_ROOT
WSA_FLAG_MULTIPOINT_D_LEAF
```

第一个标志 WSA_FLAG_OVERLAPPED，用于指定这个套接字具备重叠 I/O (是适用于 Winsock 的可能实现的通信模式之一)。这个主题将在第 8 章详细讨论。调用 socket 建立一个套接字时，WSA_FLAG_OVERLAPPED 便是默认设置。一般说来，在使用 WSASocket 时，最好始终保持设定该标志。后面四个标志用于处理多播套接字。

原始套接字

利用 WSASocket 建立套接字时，可向函数调用传送一个 WSAPROTOCOL_INFO 结构，以定义准备建立的那个套接字的类型；尽管如此，还是可建立一些套接字类型 (在传输提供者目录中，它们没有相应的条目)。最佳示例是 IP 协议下的原始套接字。原始套接字一种通信，允许你把其他协议封装在 UDP 数据包中，比如说“互联网控制消息协议”(ICMP)。ICMP 的目的是投递互联网主机间的控制、错误和信息型消息。由于 ICMP 不提供任何数据传输功能，因此不能把它与 UDP 或 TCP 同等看待，但它和 IP 本身属于同一个级别。第 13 章将详细论述原始套接字。

5.5 具体平台的问题

已发布的 Windows 95 支持 Winsock 1.1 规格。微软已完成了一个免费的 Winsock 更新，可从他们的 Web 站点下载 ([http://www.microsoft.com/Windows 95/downloads/](http://www.microsoft.com/Windows%2095/downloads/))。另外，Winsock 2 SDK 很有用，其中包括了编译 Winsock 2 应用程序所需要的头和库。自然，Windows 98、Windows NT 4 和 Windows 2000 都支持 Winsock 2，无须任何必要的新增项。Windows CE 只支持 Winsock 1.1 规格。

至于不同传输提供者提供的支持，应该提到的限制极少。Windows CE只支持TCP/IP和红外线套接字。而对Windows 95和Windows 98而言，Winsock API中尚未揭示NetBIOS传输提供者（利用地址家族AF_NETBIOS的传输）。若调用WSAEnumProtocols函数，不会相应地列出NetBIOS提供者，即使机器上已安装了这些NetBIOS提供者。尽管如此，通过利用原始NetBIOS接口，我们仍然可以使用NetBIOS，就象第1章所说的那样。最后，列出的RSVP（由它提供QoS）和ATM提供者当然可用于Windows 98和Windows 2000。

Winsock API和OSI模型

现在，我们来看看本章提及的一些概念是如何与OSI模型关联的（参见第4页上的图1-1）。Winsock目录（通过WSAEnumProtocols列举出来的）中的传输提供者位于OSI模型的传送层。也就是说，每个传输协议都会提供一种传输数据的方法；但是，它们本身又是另一个网络协议的成员，而网络协议位于网络层，因为它是为网络上各节点提供定址方法的协议。比如，UDP和TCP就是传输协议，但两者又都属于因特网协议（IP）。

Winsock API安装在“会话层”和“传送层”之间。Winsock提供了一种可为指定传输协议打开、计算和关闭会话的能力。在Windows下，上面三层（应用层、呈现层和会话层）在很大程度上与用户的Winsock应用有关。换言之，用户的Winsock应用控制了会话的方方面面，必要时，还会根据程序的需要格式化数据。

5.6 选择适当的协议

在开发网络应用程序时，可从现有的大量协议中选出自己需要的，用来作为应用程序的基础。如果应用程序需要依靠特定协议进行通信，选择余地就有限；但是，如果想从头开发一个应用程序，TCP/IP就是首选协议之一，至少从支持能力和微软的赞助这一角度来看，是这样的。AppleTalk、NetBIOS和IPX/SPX都囊括在微软的操作系统中，以便提供与其他操作系统的兼容性，同时，它们也是网络编程的重要元素。由此可见，在网络计算机上安装Windows 95时，默认安装的协议便是NetBEUI和IPX/SPX。

随着互联网的爆炸性增长，许多大公司、教育部门以及其他部门都把TCP/IP选作为自己的通信协议。Windows 2000中，微软还专门强调了TCP/IP。现在，大多数网络服务都以这个协议为基础，从而降低了对NetBIOS的依赖性。除此以外，在微软的TCP/IP实施方案中找到错误时，马上可以得到错误修复的响应，反之，其他协议中的错误要得到修复的话，可能就不那么幸运了（也许还是），这要看具体情况。

也就是说，在选择协议时，如果想把它用于网络应用，TCP/IP是你最安全的选择。除此以外，微软一直对ATM网络提供强大的技术支持。如果有闲开发只运行于ATM网络的应用程序，就应该在Winsock利用原始ATM，这样兴许比较安全。当然，TCP/IP用户应该注意到ATM网络可配置成运行于TCP/IP仿真模式，而且在这一模式中，TCP/IP同样可以正常运行。除了开发任何应用程序都需要的协议特性之外，这些只是需要考虑到少数因素。

5.7 小结

通过本章的学习，大家已了解为应用程序选择网络传输时应该知道的基本特性。在为指

定的协议开发成功的网络应用程序时，了解这些特性是至关重要的。我们还有计划地深入探讨了如何获得安装在系统中的传输提供者列表和如何查询特定属性。最后，我们还学习了如何为指定的传输建立套接字，方法是为 `WSASocket` 或 `socket` 函数指定正确的参数，再利用 `WSAEnumProtocols` 查询目录条目以及通过 `WSAPROTOCOL_INFO` 结构，把函数投递到 `WSASocket`。下一章，我们将进一步探讨各主要协议的定址方法。