第9章　套接字选项和I/O控制命令

套接字一旦建立，通过套接字选项和 I/O控制命令对各种属性进行操作，便可对套接字的行为产生影响。有的选项只用于信息的返回，而有的选项则可在应用程序中影响套接字的行为。 I/O控制命令肯定会对套接字的行为产生影响。本章着重讨论四个 Winsock函数：getsockopt、setsocketopt、ioctlsocket和WSAIoctl。每个函数都有大量命令，其中大多数尚未正式或正确写入参考文档。本章将讨论各函数需要的参数和可以使用的选项，以及哪些平台对这些选项提供了支持。在此，我们假定各个选项都能在 Win32平台上运行（Windows CE、Windows 95、Windows 98、Windows NT和Windows 2000）；如有例外，我们会专门注明。但选项在要求Winsock 2的支持时，情况可能有所不同。因为 Winsock 2本身便不能在所有平台上通用。因此， Winsock 2的I/O控制命令和选项在Windows CE与Windows 95平台上未获支持（除非已在Windows 95上安装了Winsock 2升级补丁程序）。另外还要记住：Windows CE不支持除TCP/IP以外的其他协议的任何专用选项。

这些I/O控制命令和选项大多定义在Winsock.h或Winsock2.h内，具体取决于它们到底从属于Winsock 1，还是从属于Winsock 2。但是，也有少数几个选项是 Microsoft提供者或某种传输协议所特有的。微软特有的一些扩展定义在 Winsock.h和Mswsock.h这两个头文件内。而传输提供者扩展定义在与其协议对应的头文件内。针对那些传输特有选项，我们将随选项一道，指明正确的头文件是什么。要注意的是，若应用程序使用了微软特有的扩展，那么必须同Mswock.lib建立链接。

9.1　套接字选项

对getsocketopt（获得套接字选项）函数来说，它的常见用法是获得与指定套接字相关的信息。其原型如下：

```
int getsockopt (
    SOCKET s,
    int level,
    int optname,
    char FAR* optval,
    int FAR* optlen
);
```

第一个参数s指定的是一个套接字，我们打算在这个套接字上执行指定的选项。对你打算使用的具体协议来说，这个套接字必须是有效的。大多数选项都是一种特定的协议和套接字类型专有的，而其他选项适用于所有类型的套接字（特别是第二个参数 level）。SOL_SOCKET的一个选项级别表明它是一个通用选项，并不一定要与一种给定的协议有关。但之所以说“不一定”，是由于并非所有协议都实现了级别SOL_SOCKET定义的每一个套接字选项。例如，SO_BROADCAST可将一个套接字置入广播模式，但并非所有支持的协议都支持广播套接字的概念。 optname参数是我们在此真正感兴趣的选项。这些选项名均是在 Winsock头文件内定义的常数值。最常见的与协议无关选项（比如和 SOL_SOCKET级别关联在一起的选项）是在

Winsock.h和Winsock2.h这两个头文件中定义的。对于每种特定的协议来说，它们都有自己的头文件，定义了与之对应的特定选项。最后， optval和optlen参数是两个变量，用于返回目标选项的值。大多数情况下，选项值都是一个整数（但也不是绝对的）。

setsocketopt函数用于在一个套接字级别或由协议决定的级别上设置套接字选项。 它的定义如下：

```
int setsockopt (
    SOCKET s,
    int level,
    int optname,
    const char FAR * optval,
    int optlen
);
```

它的参数和getsocketopt函数的参数相同，例外的是我们以 optval和optlen参数的形式，将值传递进去。这些值是为指定的选项设定的。和 getsocketopt函数一样， optval大多数时候都是一个整数，但也并非总是如此。正式编程的时候，应查询对每个选项的说明，了解到底该将什么作为选项值传递进去。

调用getsocketopt或setsocketopt时，最常见的错误是试图获得一个套接字的信息，但那个套接字的基层协议却不具备某种指定的特征（或选项）。例如，类型为SOCK_STREAM的一个套接字本身是不能对数据进行广播操作的；因此，若试图设置或获取 SO_BROADCAST选项，便会造成WSAENOPROTOOPT错误。

9.1.1 SOL_SOCKET选项级别

本节介绍可根据套接字本身的特征，返回信息的一些套接字选项，但这些信息并非那个套接字的基层协议所特有的（与它无关）。

1. SO_ACCEPTCONN

选项值类型	获取/设置	Winsock版本	说明
布尔值	只能获取	1+	如为TRUE（真），表明套接字处于监听模式

如果已通过 Listen函数，将套接字置入监听模式，这个选项就会返回 TRUE。SOCK_DGRAM类型的套接字不支持这一选项。

2. SO_BROADCAST

选项值类型	获取/设置	Winsock版本	说明
布尔值	两种均可	1+	如为TRUE，表明套接字已配置成对广播消息进行发送

如果指定的套接字已经配置成收发广播数据，对这个套接字选项进行查询，就会返回TRUE。随SO_BROADCAST一起使用setsockopt，便可在这个套接字上启用广播通信功能。对于非SOCK_STREAM类型的所有套接字来说，这个选项都是有效的。

广播是一种特殊的数据发送方法，使本地子网上的所有机器都能收到相同的数据。当然，在监听进入广播数据的机器上，必须进行某些形式的处理。广播通信的缺点在于假如同时有许多进程发送广播数据，网络立刻就会拥挤不堪，造成网络性能大打折扣，甚至有可能陷入瘫痪。要想接收一条广播消息，首先必须启用广播选项，然后使用某个数据报接收函数，比如recvfrom或WSARecvfrom。另外还有一种方法，即把套接字与广播地址连接起来，这是通过调用connect或WSAConnect，再调用recv或WSARecv来实现的。对UDP广播来说，必须指定一个端口号，以便向它发送数据报；类似的，接收端也必须请求在这个端口上接收广播数

据。下面的示范代码解释了如何通过UDP发送广播数据：

```
SOCKET        s;
BOOL          bBroadcast;
char          *sMsg = "This is a test";
SOCKADDR_IN bcast;

s = WSASocket(AF_INET, SOCK_DGRAM, 0, NULL, 0, WSA_FLAG_OVERLAPPED);
bBroadcast = TRUE;
setsockopt(s, SOL_SOCKET, SO_BROADCAST, (char *)&bBroadcast,
    sizeof(BOOL));
bcast.sin_family = AF_INET;
bcast.sin_addr.s_addr = inet_addr(INADDR_BROADCAST);
bcast.sin_port = htons(5150);
sendto(s, sMsg, strlen(sMsg), 0, (SOCKADDR *)&bcast, sizeof(bcast));
```

对UDP来说，存在着一个特殊的广播地址，所有广播数据均应发至该地址。这个地址便是255.255.255.255。通过为INADDR_BROADCAST提供一个#define引导命令，可使整个程序更为简单，而且更加易读。

AppleTalk是能够发送广播消息的另一种协议。AppleTalk也提供了一个特殊地址，由广播数据专用。通过第6章的学习，大家已经知道AppleTalk地址共由三部分组成：网络、节点和套接字（目的地）。要进行广播通信，将目的地设为ATADDR_BROADCAST(0xFF)，这样便会促使数据报发给一个指定网络内的所有端点。

通常，只有在发送广播数据报时，才需要设置SO_BROADCAST选项。要想接收一个广播数据报，只需在那个指定的端口上，对进入的数据报进行监听即可。但在Windows 95操作系统中，在使用IPX协议的时候，接收套接字必须设置SO_BROADCAST选项，以便接收广播数据。这一点已在“微软知识库”编号为Q137914的文章里得到了特别声明，地址是http://support.microsoft.com/support/search。这是Windows 95的一个错误。

3. SO_CONNECT_TIME

选项值类型	获取/设置	Winsock版本	说明
整数	只能获取	1+	返回套接字建立连接的时间，以秒为单位

SO_CONNECT_TIME是一个微软专用选项，用于返回连接建立的秒数。该选项最常见的用法是和AcceptEx函数一道使用。AcceptEx要求为进入的客户机连接请求，传递一个有效的套接字句柄。该选项可在客户机的SOCKET句柄上调用，以判断连接是否已经建立，以及建立了多久的时间。若套接字当前尚未连接，返回值便是0xFFFFFFFF。

4. SO_DEBUG

选项值类型	获取/设置	Winsock版本	说明
布尔值	两者均可	1+	如果是TRUE，就允许调试输出

推荐Winsock服务提供者在应用程序设置了SO_DEBUG选项的前提下，输出调试信息。至于调试信息如何展示，要取决于服务提供者的基层实施方式。要想打开调试信息，可设置SO_DEBUG，将布尔变量设为TRUE，调用setsockopt函数。若随SO_DEBUG调用getsockopt，会返回TRUE或FALSE，分别代表允许和禁止调试。不幸的是，目前尚无任何一种Win32平台支持SO_DEBUG选项，这在“微软知识库”编号为Q138965的文章里已有说明。尽管在设置了该选项的前提下，不会返回任何错误，但基层网络提供者会将其忽略。

5. SO_DONTLINGER

选项值类型	获取/设置	Winsock版本	说明
布尔值	两者均可	1+	如果是TRUE，则禁用SO_LINGER

某些协议支持套接字连接的“安全”关闭。对这些协议来说，它们实现了一种特殊的机制，可防止在通信一方或双方关闭了套接字的前提下，造成尚未处理或尚未传输完毕的数据的丢失。要想改变这种行为，可设置SO_LINGER选项，然后调用setsockopt函数，从而改变这种行为。经过一段规定的时间后，套接字及其所有资源都被强行清除。与套接字关联在一起的所有待决或未传输完毕的数据都会被丢弃，同时重设与对方的连接（WSAECONNRESET）。若设置了SO_DONTLINGER（不拖延）选项，可确保不经历这样的一段拖延时间。若随SO_DONTLINGER一起调用getsockopt，会返回一个布尔类型的TRUE或FALSE值，分别表示设置或没有设置一个拖延时间值。若在设置了SO_DONTLINGER的前提下，调用setsockopt，便会将这种拖延禁止。要注意的是，类型为SOCK_DGRAM的套接字不支持这一选项。

6. SO_DONTROUTE

选项值类型	获取/设置	Winsock版本	说明
布尔值	两者均可	1+	如果是TRUE，便直接向网络接口发送消息，毋需查询路由表

SO_DONTROUTE（不路由）选项用于指示位于基层的网络堆栈，忽略路由表的存在，通过套接字绑定的那个接口直接将数据传送出去。例如，假定我们创建了一个UDP套接字，并将其与接口A绑定到一起，然后发送一个数据包，目的地是同接口B连接的另一个网络上的某台机器。此时，若采用默认设置，该数据包会经过一个路由过程，使其能通过接口B传入目标网络。但将这里的布尔值设为TRUE，再来调用setsockopt，便可禁止这样做，数据包会从绑定的接口上直接发送出去。以后可调用getsockopt，判断是否允许了路由。要注意的是，在默认情况下，路由是允许的。

可在一个Win32平台上成功调用该选项。但是，Microsoft提供者会悄悄地忽略这一请求，并总是通过路由表为外出数据决定一个适当的接口。

7. SO_ERROR

选项值类型	获取/设置	Winsock版本	说明
布尔值	只能获得	1+	返回错误状态

SO_ERROR选项用于返回和重设以具体套接字为基础的错误代码。这些错误与那些以具体线程为基础的错误代码颇有不同，后者是用WSAGetLastError和WSASetLastError函数调用来控制的。若使用套接字，进行了一次成功的调用，那么不会重设由SO_ERROR选项返回的、以具体套接字为基础的错误代码。尽管调用该选项不会造成错误；但是，错误值并非肯定能够及时更新。因此，该选项有些时候会返回0值（表明没有错误）。除非绝对需要依赖单独的错误代码，否则最好还是使用WSAGetLastError。

8. SO_EXCLUSIVEADDRUSE

选项值类型	获取/设置	Winsock版本	说明
布尔值	两者均可	2+	如果是TRUE，套接字绑定的那个本地端口就不能重新被另一个进程使用

该选项是对SO_REUSEADDR（重复使用地址）的一个补充，后者很快就会讲到。该选项的作用是禁止其他进程在一个本地地址上使用SO_REUSEADDR（我们的应用程序正在这个地址上运行）。例如，假定两个单独的进程都与同一个本地地址绑定到了一起（假定早先已设置了SO_REUSEADDR），那么两个套接字中到底由哪一个负责接收进入连接请求通知呢？

这一点并未定义！如果应用程序的要求非常苛刻（用于关键性任务的程序），这一点显然是非常不幸的。SO_EXCLUSIVEADDRUSE选项的作用便是将一个本地地址牢牢锁定在与它绑定的那个套接字上。这样一来，假如有其他进程试图针对相同的本地地址使用SO_REUSEADDR，进程调用便会失败。若想设置该选项，必须具有“管理员”的身份。而且要注意的是，它适用于Windows 2000！

9. SO_KEEPALIVE

选项值类型	获取/设置	Winsock版本	说明
布尔值	两者均可	1+	如果是TRUE，套接字就会进行配置，在会话过程中发送“保持活动”消息

对一个以TCP为基础的套接字来说，应用程序可请求基层服务提供者允许在TCP连接上使用“保持活动”（Keep Alive）数据包，这是通过打开SO_KEEPALIVE套接字选项来做到的。在Win32平台上，“保持活动”是根据RFC 1122文件的4.2.3.6小节的规定而实现的。假如由于“保持活动”造成了连接的中断，便会向套接字上正在进行的任何一个调用返回一个WSAENETRESET错误代码。而后续的任何调用都会返回WSAENOTCONN错误。要想了解更多的实施细节，请大家参阅RFC文件。在此要注意的一个要点，“保持活动”信号的发送要以不短于2小时的时间间隔进行。2小时的“保持活动”时间是通过注册表配置的；但是，一旦改变了这个默认值，同时也会影响系统中所有TCP连接的“保持活动”行为。而这种后果通常是应当避免的。另一个办法是实现自己的“保持活动”策略。在第7章，我们已讨论了这种类型的策略。要注意的是，SOCK_DGRAM类型的套接字并不支持这一选项。

与“保持活动”设置对应的注册表键是KeepAliveInterval（保持活动间隔）和KeepAliveTime（保持活动时间）。这两个键均用来保存类型为REG_DWORD的双字值，单位是“毫秒”。其中，前一个键指定每次“保持活动”传输的间隔时间，直至收到一个响应。后一个键用于控制TCP以多大的频率尝试一个“保持活动”数据包的发送，以查验一个理想的连接是否仍然有效。在Windows 95及Windows 98操作系统中，这两个键都位于下述注册表路径中：

\HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\VxD\MSTCP

而在Windows NT和新出的Windows 2000中，这两个键位于下述位置：

\HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\TCPIP\Parameters

特别要指出的是，在Windows 2000操作系统中，引入了一个新的I/O控制命令：SIO_KEEPALIVE_VALS。可用它分别针对每一个套接字，更改其“保持活动”值以及发送的间隔时间。再也不用像原来那样，只能在整个系统的范围内更改。本章稍后还会对这个I/O控制命令进行详解。

10. SO_LINGER

选项值类型	获取/设置	Winsock版本	说明
Struct linger	两者均可	1+	设置或获取当前的拖延值

LINGER是“拖延”的意思。SO_LINGER用于控制在未发送的数据排队等候于套接字上的时候，一旦执行了closesocket命令，那么该采取什么样的行动。若在设置了该选项的前提下，调用getsockopt，便会在一个linger结构中，返回当前的拖延时间设定。该结构定义如下：

```
struct linger {
    u_short l_onoff;
```

```
    u_short l_linger;
}
```

若l_onoff是一个非零值，意味着此时允许进行关闭拖延，而 l_linger对应的是一段拖延时间，以秒为单位。若超出规定的时间，便不再拖延，所以尚未发送或接收的数据都会丢弃，同时重设与对方的连接。相反，我们可调用 setsockopt，在丢弃任何正在排队等候的数据之前，启用拖延功能，同时指定一个时间长度。要想达到这个目的，需要在类型为 struct linger的一个变量中指定希望的时间长度。

若随setsockopt一道设置了拖延时间值，那么必须将结构的 l_onoff字段设为一个不为零的值（非零值）。若允许拖延，后来又想将其禁止，可随 SO_LINGER选项一道，调用setsockopt，同时将linger结构的l_onoff字段设为0。此外，亦可通过SO_DONTLINGER（不拖延）选项调用setsockopt，为其optval参数传递一个TRUE值。但要注意的是，类型为SOCK_DGRAM的套接字不运行这一选项。

调用closesocket函数的时候，对拖延选项的设置会直接影响到一个连接的行为。在表 9-1中，我们对不同的行为进行了总结。

表9-1 拖延选项

选 项	间隔时间	关闭方式	是否等待关闭？
SO_DONTLINGER	不适用	从容关闭	否
SO_LINGER	0	强行关闭	否
SO_LINGER	非零值	从容关闭	是

假如将SO_LINGER设为零（换言之，linger结构的成员l_onoff不为0，但l_linger为0），便不用担心closesocket调用会进入“锁定”状态（等待完成），即使队列中的数据尚未发送，或者尚未发出收到确认。我们称这种情况为“强行关闭”，或者“野蛮关闭”，因为套接字虚拟通信回路会立即重设，尚未发出的所有数据都会丢失。而在远程端，正在运行的任何接收调用都会失败，同时返回WSAECONNRESET错误。

如果在一个锁定套接字上，设置了 SO_LINGER，那么closesocket调用也会在一个锁定套接字上进入锁定或暂停状态，直到剩余的数据全部发出，或者超过规定的时间。我们将这称为“从容关闭”。若超过规定的时间，数据仍未全部发出， Windows套接字机制便会在closesocket返回之前，将连接中断。

11. SO_MAX_MSG_SIZE

选项值类型	获取/设置	Winsock版本	说明
无符号整数	只能获取	2+	对一个面向消息的套接字来说，一条消息的最大长度

这是一种只能获取数据的套接字选项，用于针对一个由特定服务提供者实现的、面向消息的套接字类型，设置一条消息的最大外出发送长度。而对那些面向字节流的套接字来说，该设定没有任何用处。在那些套接字上，没有办法知道最大的进入消息长度。

12. SO_OOBINLINE

选项值类型	获取/设置	Winsock版本	说明
布尔值	两者均可	1+	如果是TRUE，带外数据就会在普通数据流中返回

默认情况下，“带外”（Out-of-band，OOB）数据不是内嵌传送的。也就是说，若调用一个接收函数（同时相应地设置 MSG_OOB标志），那么OOB数据只需单独一次调用便会返回。若设置了这个选项， OOB数据会出现于自一个接收调用返回的数据流中。而在设置了

SIOCATMARK选项的前提下，若调用 ioctlsocket，便可决定哪个字节属于 OOB数据。类型为SOCK_DGRAM的套接字同样不运行这一选项。不幸的是，在目前各个版本的 Win32平台上，均不支持这个套接字选项。要想了解 OOB数据的详情，可参阅第 7章。

13. SO_PROTOCOL_INFO

选项值类型	获取/设置	Winsock版本	说明
WSAPROTOCOL_INFO	只能获得	2+	套接字绑定的那种协议的特征

这又是一个“只能获得”或者“只能用来收取数据”的选项，可在指定的WSAPROTOCOL_INFO结构中，填充与套接字关联在一起的协议的特征信息。请参考第 6章，了解WSAPROTOCOL_INFO结构的详细说明，及其各个成员字段的详情。

14. SO_RCVBUF

选项值类型	获取/设置	Winsock版本	说明
整数	两者均可	1+	面向接收操作，为每个套接字分别获取或设置缓冲区长度

这是一个非常简单的选项，用于返回或设置分配给该套接字的缓冲区大小。这个缓冲区用于数据的接收。创建好一个套接字后，会为其分配一个发送缓冲区和一个接收缓冲区，分别用于数据的发送与接收。若请求将接收缓冲区的大小设为一个特定的值，那么即便没有充分满足这个请求，没有提供全部要求的空间，对 setsockopt的调用也会成功，不会返回错误。要想确保请求的缓冲区空间都已分配，可调用 getsockopt，调查实际分配了多大的空间。目前，除Windows CE以外，所有Win32平台都能获取或设置接收缓冲区的大小。在 Windows CE中，我们不能更改这个值，只能“取得”它。

之所以要更改缓冲区的大小，一个可能的原因是需要根据应用程序的行为，对缓冲区大小进行相应地调整。例如，若编写一段代码，想进行 UDP数据报的接收，那么通常应将接收缓冲区的大小设为数据报本身长度的几倍。而对重叠 I/O来说，若将缓冲区的大小设为 0，那么在某些情况下，可能会在一定程度上提升性能：若这些缓冲区的长度不为 0，必须牵涉到额外的内存复制操作，以便将数据从系统缓冲区移至由用户指定的缓冲区。假如没有中间缓冲区，数据就会立即复制到用户指定的缓冲区内。而这样的的操作只有在同时明确进行多个接收调用的前提下，才显得有效。假如只进行一次接收，可能会对性能造成严重影响，因为除非指定一个缓冲区，而且准备好接收数据，否则本地系统是不会接受任何进入数据的。欲知详情，请参考第8章8.2.5节“完成端口模型”中“其他问题”小节。

15. SO_REUSEADDR

选项值类型	获取/设置	Winsock版本	说明
布尔值	两者均可	1+	如果是TRUE，套接字就可与一个正由其他套接字使用的地址绑定到一起，或与处在TIME_WAIT状态的地址绑定到一起

默认情况下，套接字不同一个正在使用的本地地址绑定到一起。但在少数情况下，仍有必要以这种方式，来实现对一个地址的重复利用。通过第 7章的学习，大家已经知道，每个连接都是通过它的本地及远程地址的组合，“独一无二”地标识出来的。针对我们想要连接的地址，只要能用极其细微的差异（比如 TCP/IP中采用不同的端口号），来维持这种“独一无二”或者“唯一”的特点，绑定便是允许的。

唯一例外的是监听套接字。两个独立的套接字不可与同一个本地接口（在 TCP/IP的情况下，则是端口）绑定到一起，以等待进入的连接通知。假定两个套接字都在同一个端口上进行监听，那么到底由谁来接收一个进入连接通知呢？对于这个问题，目前尚无一种正式规范提出了解决方案。在 TCP的环境下，假如服务器关闭，或异常退出，造成本地地址和端口均

进入TIME_WAIT状态，那么SO_REUSEADDR这个套接字选项便显得非常有用。在TIME_WAIT状态下，其他任何套接字都不能与那个端口绑定到一起。但假若设置了该选项，服务器便可在重新启动之后，在相同的本地接口及端口上进行监听。

16. SO_SNDBUF

选项值类型	获取/设置	Winsock版本	说明
布尔值	两者均可	1+	如果是TRUE（非零值），意味着套接字被配置成可进行广播消息的发送

这也是一个非常简单的选项，要么返回、要么设置分配给套接字的数据发送缓冲区的大小。创建好一个套接字后，会为其分配一个发送缓冲区和一个接收缓冲区，分别用于数据的发送及接收。若请求将发送缓冲区的大小设为一个特定的值，那么即便没有充分满足这个请求（没有提供要求的全部空间），对setsockopt的调用也会成功，不会返回错误信息。但是，假如希望确定请求的缓冲区空间都已正确分配，可调用 getsockopt，调查目前实际分配了多大的空间。目前，除Windows CE以外，所有Win32平台都能获取或设置发送缓冲区的大小。在Windows CE中，我们不可更改这个值，但能“取得”它。

和SO_RCVBUF一样，可用SO_SNDBUF选项将发送缓冲区的大小设为0。对于锁定发送调用来说，将缓冲区的大小设为0后，一个好处在于：一旦调用完成，便可肯定自己的数据正在传输途中。此外，和在接收操作中将缓冲区的长度设为 0一样，不会涉及数据向系统缓冲区进行的额外内存复制操作。但是，假如发送缓冲区的长度不为 0，那么通过默认的堆栈缓冲机制，可充分发挥数据流水线的优势。将这个长度变成 0后，会失去这一优势，所以这是它的一个缺点。换言之，在默认情况下，假如要连续不停地执行数据的发送操作，那么本地网络堆栈可将我们的数据复制到一个系统缓冲区，在以后某个恰当的时间发送出去（取决于当时采用的I/O模型）。而另一方面，假如应用程序需要的其他方面的优势，那么将发送缓冲区屏蔽后，亦可省下进行内存复制时的一些机器指令开销。欲知这方面的详情，可参阅第 8章8.2.5节“完成端口模型”中“其他问题”小节。

17. SO_TYPE

选项值类型	获取/设置	Winsock版本	说明
整数	只能获取	1+	返回指定套接字的类型（如 SOCK_DGRAM和SOCK_STREAM等等）

SO_TYPE选项是一个只能用来“获取”数据的选项，用于返回指定套接字的类型。可能的套接字类型包括SOCK_DGRAM、SOCK_STREAM、SOCK_SEQPACKET、SOCK_RDM和SOCK_RAW等等。

18. SO_SNDTIMEO

选项值类型	获取/设置	Winsock版本	说明
整数	两者均可	1+	获取或设置套接字上的数据发送超时时间（以毫秒为单位）

SO_SNDTIMEO选项用于在一个锁定套接字上调用Winsock发送函数时，设定一个“超时”值。这是一个以毫秒为单位的整数值，设定发送函数最多执行多久。假如需要使用SO_SNDTIMEO选项，并用WSASocket函数创建套接字，那么必须将WSA_FLAG_OVERLAPPED设为WSASocket的dwFlags参数的一部分。以后对任何Winsock发送函数的调用（send、sendto、WSASend、WSASendTo等等）都只会锁定指定数量的时间。如发送操作在那段时间内不能完成，调用便会失败，并返回错误10060（WSAETIMEDOUT）。

考虑到性能方面的原因，这个选项在Windows CE 2.1中已被禁止。如试图设置这个选项，

那么它会被简单地忽略，而且不会返回任何错误提示。但以前版本的 Windows CE却实现了这一选项。

19. SO_RCVTIMEO

选项值类型	获取/设置	Winsock版本	说明
整数	两者均可	1+	获取或设置与套接字上数据接收对应的超时时间值（以毫秒为单位）

SO_RCVTIMEO选项用于设置一个锁定套接字上的接收超时值。这是一个以毫秒为单位的整数，用于指出一个Winsock接收函数在接收数据时，最多应“锁定”多长时间。如需使用SO_RCVTIMEO选项，并用 WSASocket函数创建套接字，那么必须将 WSA_FLAG_OVERLAPPED指定成为WSASocket的dwFlags参数的一部分。以后对任何 Winsock接收函数的调用（包括recv、recvfrom、WSARecv、WSARecvFrom等等）都只会锁定指定的时间长度。假如在那段时间内，没有数据抵达，调用便会失败，并返回错误 10060（WSAETIMEDOUT）。

考虑到性能方面的因素，这一选项在 Windows CE 2.1中已被禁止。如试图设置它，会被简单地忽略，而且不会返回任何错误提示。但以前版本的 Windows CE却实现了这一选项。

20. SO_UPDATE_ACCEPT_CONTEXT

选项值类型	获取/设置	Winsock版本	说明
SOCKET	两者均可	1+	获取或设置与套接字上的数据接收对应的超时值（以毫秒计）

该选项属于Microsoft特有的一项扩展，经常与 AcceptEx函数联合使用。该函数最显著的一个特点在于，它属于Winsock 1规范的一部分，允许通过重叠I/O操作，来进行数据接收调用的执行。该函数要将监听套接字设为自己的一个参数，同时还要指定一个相应的套接字句柄，它会成为一个已接受的客户机。为使监听套接字的特征能完整转移至客户机套接字，必须设置这个套接字选项。对于提供“服务质量”（QoS）功能的监听套接字来说，这一点尤为重要。要想使客户机套接字具有保障 QoS的功能，这个选项是必须设置的。为在一个套接字上设置该选项，可将监听套接字作为 setsockopt的SOCKET参数，并以optval的形式，传递接受套接字句柄（比如客户机句柄）。要注意的是，该选项仅适用于 Windows NT和Windows 2000这两种操作系统。

9.1.2 SOL_APPLETALK选项级别

下述选项均为AppleTalk协议之专用套接字选项，只能用于通过 socket或WSASocket函数创建的套接字（同时设置 AF_APPLETALK标志）。这里列出的大多数选项都与 AppleTalk名字的设置或获取有关。欲了解 AppleTalk地址家族更详细的信息，可参考第 6章的内容。某些AppleTalk套接字选项（比如 SO_DEREGISTER_NAME）提供了不止一个的选项名。在这种情况下，所有选项的名字均可互换使用。

1. SO_CONFIRM_NAME

选项值类型	获取/设置	Winsock版本	说明
WSH_NBP_TUPLE	只能获取	1	确定指定的AppleTalk名字和指定地址绑定在一起

SO_CONFIRM_NAME选项用于检验一个指定的 AppleTalk名字是否已和指定的地址绑定到了一起。这样可促使向目标地址发送一个“名字绑定协议”（NBP）检索请求，以校验指定的名字。若此次调用以失败告终，并返回了一个 WSAEADDRNOTAVAIL错误，便表明名字尚未与指定的地址绑定到一起。

2. SO_DEREGISTER_NAME，SO_REMOVE_NAME

选项值类型	获取/设置	Winsock版本	说明
WSH_REGISTER_NAME	只能设置	1	从网络中撤消对指定名字的注册

该选项用于将一个名字从网络中撤消对它的注册。若指定的名字目前并未存在于网络中，调用返回后，仍会注明“操作成功”。请参阅第6章的6.5.2节“AppleTalk名的注册”小节，了解WSH_REGISTER_NAME结构的详情，它实际是WSH_NBP_NAME结构的另一个名字。

3. SO_LOOKUP_MYZONE，SO_GETMYZONE

选项值类型	获取/设置	Winsock版本	说明
字符	只能获取	1	返回网络上的默认网区

该选项可返回网络上的默认“网区”。getsockopt调用的optval参数应当是一个字串，长度至少33个字符。要注意的是，NBP名字的最大长度是由MAX_ENTITY_LEN规定的，后者定义成32。若实际的名字长度不够，便需加上额外的字符，形成“空终止符”。

4. SO_LOOKUP_NAME

选项值类型	获取/设置	Winsock版本	说明
WSH_LOOKUP_NAME	只能获取	1	查找指定的NBP名字，并返回相符的名字及NBP信息字元组

该选项用于在网络上查找一个指定的名字（比如，在一个客户机想建立与服务器的连接时）。连接建立之前，字面上易于理解的英语名字必须解析成一个具体的AppleTalk地址。大家可参考第6章6.5.3节“AppleTalk名的解析”，那里提供了一段示范代码，可直接用来进行AppleTalk名字的解析。

在此要注意的一件事情是，一旦成功返回，WSH_NBP_TUPLE结构便会占据指定缓冲区中WSH_LOOKUP_NAME信息之后的空间。换言之，当初调用getsockopt函数的时候，便应提供一个足够大的缓冲区，以便在缓冲区的开头，完全装下WSH_LOOKUP_NAME信息，并在剩下的空间里，装下大量WSH_NBP_TUPLE结构。图9-1向大家展示了在调用之初，如何准备好这个缓冲区（首先容纳的是WSH_LOOKUP_NAME结构）；以及在返回后，WSH_NBP_TUPLE结构放在什么位置。

图9-1 SO_LOKUP_NAME缓冲区

5. SO_LOOKUP_ZONES，SO_GETZONELIST

选项值类型	获取/设置	Winsock版本	说明
WSH_LOOKUP_ZONES	只能获取	1	返回来自Internet网区列表的区名

该选项要求提供一个足够大的缓冲区，以便在头部位置容下一个WSH_LOOKUP_ZONES结构。成功返回之后，在WSH_LOOKUP_ZONES结构之后的空间里，便会包含一系列“空终止”的区名列表。下述代码向大家解释了如何利用这个SO_LOOKUP_ZONES选项：

```
PWSH_LOOKUP_NAME    atlookup;
PWSH_LOOKUP_ZONES   zonelookup;
char                cLookupBuffer[4096],
                    *pTupleBuffer = NULL;

atlookup = (PWSH_LOOKUP_NAME)cLookupBuffer;
zonelookup = (PWSH_LOOKUP_ZONES)cLookupBuffer;
ret = getsockopt(s, SOL_APPLETALK, SO_LOOKUP_ZONES, (char *)atlookup,
```

```
    &dwSize);
pTupleBuffer = (char *)cLookupBuffer + sizeof(WSH_LOOKUP_ZONES);
for(i = 0; i < zonelookup->NoZones; i++)
{
    printf("%3d: '%s'\n", i + 1, pTupleBuffer);
    while (*pTupleBuffer++);
}
```

6. SO_LOOKUP_ZONES_ON_ADAPTER，SO_GETLOCALZONES

选项值类型	获取/设置	Winsock版本	说明
WSH_LOOKUP_ZONES	只能获取	1	为指定的适配器名返回一个区名列表

该选项类似于SO_LOOKUP_ZONES，只是要求我们先指定一个适配器名，再来取得与那个适配器连接的网络对应的本地区名列表。同样地，在此必须提供一个足够大的缓冲区，在其头部装下一个WSH_LOOKUP_ZONES结构。而返回的空终止区名列表自 WSH_LOOKUP_ZONES结构之后开始存放。此外，适配器的名字必须以一个 UNICODE字串（WCHAR）的形式传递。

7. SO_LOOKUP_NETDEF_ON_ADAPTER，SO_GETNETINFO

选项值类型	获取/设置	Winsock版本	说明
WSH_LOOKUP_NETDEF_ON_ADAPTER	只能设置	1	为指定网络返回种子值，以及默认网区

该选项可为指定的网络编号返回其种子值；同时返回一个以“空”终止的ANSI字串，其中包含了与指定适配器连接的那个网络的默认网区。适配器是在结构之后，以一个 UNICODE（WCHAR）字串的形式传递的，并会由函数返回的默认网区覆盖。假如网络并未进行“种子”设定，那么会返回一个网络地址范围1~0xFFFE，同时在空终止的ANSI字串中包含默认网区“*”。

8. SO_PAP_GET_SERVER_STATUS

选项值类型	获取/设置	Winsock版本	说明
WSH_PAP_GET_SERVER_STATUS	只能获取	1	自指定服务器返回PAP状态

该选项可取得在ServerAddr指定之地址上注册的“打印机访问协议”（PAP）状态。而那个地址通常是通过一次NBP检索操作获得的。四个保留的字节分别对应于 PAP状态包中的四个保留字节，均按网络字节顺序排布。而一个 PAP状态字串可以是任意的，通过 SO_PAP_SET_SERVER_STATUS选项进行设置；详情还会在本章后面进行解释。下面是 WSH_PAP_GET_SERVER_STATUS结构的定义：

```
#define    MAX_PAP_STATUS_SIZE        255
#define    PAP_UNUSED_STATUS_BYTES        4

typedef struct _WSH_PAP_GET_SERVER_STATUS
{
    SOCKADDR_AT            ServerAddr;
    UCHAR                  Reserved[PAP_UNUSED_STATUS_BYTES];
    UCHAR                  ServerStatus[MAX_PAP_STATUS_SIZE + 1];

} WSH_PAP_GET_SERVER_STATUS, *PWSH_PAP_GET_SERVER_STATUS;
```

下述代码片断向大家展示了如何对 PAP状态进行请求。状态字串的长度是由 ServerStatus字段的第一个字节指定的：

```
WSH_PAP_GET_SERVER_STATUS  status;
int                        nSize = sizeof(status);
```

```
status.ServerAddr.sat_family = AF_APPLETALK;
ret = getsockopt(s, SOL_APPLETALK, SO_PAP_GET_SERVER_STATUS,
    (char *)&status, &nSize);
```

9. SO_PAP_PRIME_READ

选项值类型	获取/设置	Winsock版本	说明
char []	只能设置	1	这个调用会在一个PAP连接上填充一次读取操作，以便发送者能实际地发出数据

若在设置成PAP连接的一个套接字上调用这个选项，远程客户机便会在本地应用尚未调用recv或WSARecvEx的前提下，开始数据的发送。设置了该选项后，应用程序会在一次select调用中进入“锁定”或“暂停”状态，然后进行实际的数据读取操作。该调用的optval参数指定的是一个缓冲区，用于数据的接收。该缓冲区的长度至少应为MIN_PAP_READ_BUF_SIZE（4096）个字节。通过这一选项，可在“由读驱动”的PAP协议上，实现对非锁定套接字的支持。但要注意的是，针对自己想要读取的每个缓冲区，都必须在设置了SO_PAP_PRIME_READ选项的前提下，执行一次setsockopt调用。

10. SO_PAP_SET_SERVER_STATUS

选项值类型	获取/设置	Winsock版本	说明
char []	只能设置	1	假如另一个客户机请求状态，则设置要发送出去的状态

客户机可通过SO_PAP_GET_SERVER_STATUS，请求取得PAP状态。可用该选项对状态进行设置，以便在客户机请求状态的前提下，提交给设置命令的缓冲区能在获取命令中返回。这里的“状态”实际是一个最多255字节的缓冲区，其中包含了对应的那个套接字的状态信息。若在提供一个空缓冲区的前提下调用设置选项，则会将以前的状态值清除掉。

11. SO_REGISTER_NAME

选项值类型	获取/设置	Winsock版本	说明
WSH_REGISTER_NAME	只能设置	1	在AppleTalk网络上注册指定的名字

该选项用于在AppleTalk网络中注册一个指定的名字。若那个名字已在网络中存在，便会返回WSAEADDRINUSE错误。大家可参考第6章，那里提供了对WSH_REGISTER_NAME结构的详细说明。

9.1.3 SOL_IRLMP选项级别

SOL_IRLMP级别与IrDA（Infrared Data Association，红外线数据联盟）协议有着密切的联系，其地址家族为AF_IRDA。使用IrDA套接字选项时，要记住的一个要点在于，在不同平台上，红外线套接字的具体实施方式是有所区别的。由于Windows CE是最早支持红外线通信的一个平台，所以没有包括后来在Windows 98和Windows 2000中引入的一些新选项。在本节，我们除了对每个选项的含义进行解释外，也指明了它适用的平台。

1. IRLMP_9WIRE_MODE

选项值类型	获取/设置	Winsock版本	说明
布尔值	两者均可	1+	获取或设置IP头内的IP选项

这是另一个非常罕见的选项，用于通过IrCOMM（红外线通信端口），建立与Windows 98的通信。它的工作级别要比IrSock正常的级别稍低一些。在9线模式下，每个TinyTP或IrLMP包都包含了一个附加的1字节IrCOMM头。要想通过套接字接口做到这一点，首先需要使用IRLMP_SEND_PDU_LEN选项，取得一个IrLMP包的最大PDU长度。随后，在建立连接，或在接受一个连接请求之前，使用setsockopt函数，将套接字置入9线通信模式。这样便可告诉

堆栈为外出（即发送出去）的每个数据帧都添加一个长度为1字节的IrCOMM头（总是设为0）。每个send的长度都必须小于允许的最大 PDU长度，以便为新增的 IrCOMM字节腾出空间。IrCOMM更深入的一些技术细节已超出了本书的范围，所以在此不再赘述。该选项仅适用于Windows 98（包括第二版）和Windows 2000。

2. IRLMP_ENUMDEVICES

选项值类型	获取/设置	Winsock版本	说明
DEVICELIST	只能获得	1+	针对范围内具有红外线通信能力的设备，返回一个 IrDA设备ID列表

由红外线通信的本质使然，具有这种能力的所有设备都注定是“机动”的，或者说是“可移动”的。换言之，它们可能移至有效通信范围之外。这个选项的作用便是对有效范围之内IR设备进行查询。要想建立与另一个设备的连接，首先必须执行这一步操作，以取得自己想连接的每一个设备的设备ID。

在支持IrSock这种套接字的各种不同的平台上， DEVIDELIST结构也有所差异。这是由于新问世的一些平台除增加了这方面的支持以外，也引入了一些新的功能。前面已经说过，最早提供红外线通信能力的是 Windows CE，而后来问世的Windows 98和Windows 2000更上一层楼，新增了一系列选项，可以更加完善地支持这种类型的通信。下面是在 Windows 98及Windows 2000操作系统中，对DEVIDELIST结构的定义：

```
typedef struct _WINDOWS_DEVICELIST
{
    ULONG                       numDevice;
    WINDOWS_IRDA_DEVICE_INFO    Device[1];
} WINDOWS_DEVICELIST, *PWINDOWS_DEVICELIST, FAR *LPWINDOWS_DEVICELIST;

typedef struct _WINDOWS_IRDA_DEVICE_INFO
{
    u_char  irdaDeviceID[4];
    char      irdaDeviceName[22];
    u_char  irdaDeviceHints1;
    u_char  irdaDeviceHints2;
    u_char  irdaCharSet;
} WINDOWS_IRDA_DEVICE_INFO, *PWINDOWS_IRDA_DEVICE_INFO,
  FAR *LPWINDOWS_IRDA_DEVICE_INFO;
```

而在Windows CE中，DEVIDELIST结构是象这样定义的：

```
typedef struct _WCE_DEVICELIST
{
    ULONG                       numDevice;
    WCE_IRDA_DEVICE_INFO        Device[1];
} WCE_DEVICELIST, *PWCE_DEVICELIST;

typedef struct _WCE_IRDA_DEVICE_INFO
{
    u_char  irdaDeviceID[4];
    char    irdaDeviceName[22];
    u_char  Reserved[2];
} WCE_IRDA_DEVICE_INFO, *PWCE_IRDA_DEVICE_INFO;
```

从中可以看出，设备信息结构也发生了变化： Windows CE使用的是 WCE_IRDA_DEVICE_INFO；而Windows 98和Windows 2000使用的是WINDOWS_IRDA_DEVICE_INFO。

每个结构都包含了一个名为 irdaDeviceID的字段。这是一个长度为 4字节的标志，用于唯一性地标识出那个设备。我们需要用这个字段来填充用于连接特定设备的 SOCKADDR_IRDA结构；或者通过IRLMP_IAS_SET及IRLMP_IAS_QUERY用于，分别用来处理或获取一个“信息访问服务”（IAS）条目。

调用getsockopt函数，对红外线设备进行列举的时候， optval参数必须对应一个DEVICELIST结构。此时唯一的要求是在最开始的时候，将 numDevice字段设为0。假如没有发现任何红外线（IR）设备，那么对getsockopt的调用不会返回任何错误。完成了一次调用后，应对numbDevice字段进行检查，看看它是否大于 0。若答案是肯定的，便意味着已找到了一个或多个红外线设备。Device字段可返回与numDevice的值相等数量的一系列结构。

3. IRLMP_EXECLUSIVE_MODE

选项值类型	获取/设置	Winsock版本	说明
布尔值	两者均可	1+	如果是TRUE，表明套接字连接处于“独占”模式

该选项通常并不在应用程序中直接使用，因为它绕过了 IrDA堆栈中的TinyTP层，而直接通过IrLMP进行通信。如果真的对使用这个选项有兴趣，请查阅位于 http://www.irda.org的IrDA规范文件。该选项适用于Windows CE和Windows 2000。

4. IRLMP_IAS_QUERY

选项值类型	获取/设置	Winsock版本	说明
IAS-QUERY	只能获取	1+	在指定服务上查询IAS，并查询与其属性对应的类名

这个套接字选项是对IRLMP_IAS_SET的一个补充，用于取得与一个类名及其服务有关的信息。在发出对getsockopt的调用之前，首先必须设置好irdaDeviceID字段，将其指定为自己准备查询的那个设备的设备ID。至于irdaAttribName字段，应设为想取得具体值的一个属性字串。最常见的查询是针对LSAP-SEL号码进行的；它的属性字串是“IrDA:IrLMP:LsapSel”。接下来，需要将irdaClassName字段设为指定属性字串即将应用于的那个服务的名字。一旦填写好这些字段的内容，便可发出对 getsockopt的调用。若成功返回，在irdaAttribType字段中，会指明需要同哪个字段联合，以便从中取得信息。可根据表 9-2总结的标识符，对这个条目的含义进行诠释。最常见的返回错误是WSASERVICE_NOT_FOUND。若在指定设备上没有找到相应的服务，便会返回这样的错误。该选项适用于Windows CE，Windows 98以及Windows 2000。

5. IRLMP_IAS_SET

选项值类型	获取/设置	Winsock版本	说明
IAS_QUERY	只能设置	1+	为指定的类名和属性设置一个属性值

IAS其实是一种动态服务注册实体，可对其进行查询和修改。利用 IRLMP_IAS_SET选项，我们可在本地IAS内，为单个类设置一个属性。和 IRLMP_ENUMDEVICES一样，Windows CE采用的是一种结构，而 Windows 98和Windows 2000采用的又是另一种结构。下面是Windows 98和Windows 2000的结构定义：

```
typedef struct _WINDOWS_IAS_QUERY
{
    u_char      irdaDeviceID[4];
    char        irdaClassName[IAS_MAX_CLASSNAME];
    char        irdaAttribName[IAS_MAX_ATTRIBNAME];
    u_long      irdaAttribType;
    union
    {
        LONG    irdaAttribInt;
```

```
        struct
        {
            u_long    Len;
            u_char    OctetSeq[IAS_MAX_OCTET_STRING];
        } irdaAttribOctetSeq;
        struct
        {
            u_long    Len;
            u_long    CharSet;
            u_char    UsrStr[IAS_MAX_USER_STRING];
        } irdaAttribUsrStr;
    } irdaAttribute;
} WINDOWS_IAS_QUERY, *PWINDOWS_IAS_QUERY, FAR *LPWINDOWS_IAS_QUERY;
```

Windows CE采用的IAS查询结构如下：

```
typedef struct _WCE_IAS_QUERY
{
    u_char     irdaDeviceID[4];
    char       irdaClassName[61];
    char       irdaAttribName[61];
    u_short    irdaAttribType;
    union
    {
        int    irdaAttribInt;
        struct
        {
            int      Len;
            u_char   OctetSeq[1];
            u_char   Reserved[3];
        } irdaAttribOctetSeq;
        struct
        {
            int      Len;
            u_char   CharSet;
            u_char   UsrStr[1];
            u_char   Reserved[2];
        } irdaAttribUsrStr;
    } irdaAttribute;
} WCE_IAS_QUERY, *PWCE_IAS_QUERY;
```

在表9-2中，我们总结了irdaAttribType字段可采用的各种不同的常数，它指出了属性具体从属于哪种类型。最后两个条目不是可由我们自行设置的值，而是通过对 getsockopt的调用，同时设置一个IRLMP_IAS_QUERY套接字选项，而能在irdaAttribType字段中返回的值。之所以也在表格中列出，只是为了保证它们的完整性。

表9-2 IAS属性类型

irdaAttribType的可选值	要设置的字段
IAS_ATTRIB_INT	irdaAttribInt
IAS_ATTRIB_OCTETSEQ	irdaAttribOctetSeq
IAS_ATTRIB_STR	irdaAttribUsrStr
IAS_ATTRIB_NO_CLASS	无
IAS_ATTRIB_NO_ATTRIB	无

要想设置一个值，首先必须将irdaDevicdID设为目标红外线通信设备。在那个设备上，我

们想对其IAS条目进行修改。同样地， irdaAttribName必须设为要在上面设置属性的那个类。而irdaClassName通常对应的是一种服务，要在上面对属性进行设置。在此要记住的是，对IrSock类型的套接字来说，套接字服务器是随 IAS注册的一种服务。客户机使用与之对应的一个LSAP-SEL编号，建立与服务器的连接。 LSAP-SEL编号是与那种服务关联在一起的一种属性。要想在服务的IAS条目中修改LSAP-SEL编号，请将irdaDeviceID字段设为服务正在上面运行的那个设备的 ID。irdaAttribName字段应设为属性字串“ IrDA:IrLMP:LsapSel ”，而irdaClassName字段应设为服务的名字（比如“ MySocketServer ”）。随后，将irdaAttribType设为IAS_ATTRIB_INT，而irdaAttribInt设为新的LSAP-SEL编号。当然，更改服务的LSAP-SEL编号并不是我们推荐的一种做法，本例只是为了阐述的方便。

6. IRLMP_IRLPT_MODE

选项值类型	获取/设置	Winsock版本	说明
布尔值	两者均可	1+	若为TRUE，套接字就会配置成与具有IR能力的打印机通信

通过Winsock，可与一台具有红外线功能的打印机进行通信，向其发送需要打印的数据。要做到这一点，在正式建立连接之前，首先要将套接字置入 IRLPT模式。创建好套接字后，只需向这个选项传递一个布尔值 TRUE即可。可利用IRLMP_ENUMDEVICES选项来寻找有效通信范围之内的红外线打印机。要注意的是，有些老式打印机不能向 IAS注册自己的存在。此时，需要通过“ LSAP-SEL-xxx ”标识符，建立与它们的直接连接关系。大家可参考第 6章对IrSock的讨论，了解具体如何绕过IAS。该选项同时适用于Windows CE及Windows 2000。

7. IRLMP_SEND_PDU_LEN

选项值类型	获取/设置	Winsock版本	说明
整数	只能获取	1+	取得最大的PDU长度

使用IRLMP_9WIRE_MODE（9线模式）选项时，可通过该选项调查最大的“ 协议数据单元 ”（PDU）长度。大家可参考对IRLMP_9WIRE_MODE的说明，了解这个选项的详情，它适用于Windows CE和Windows 2000。

9.1.4 IPPROTO_IP选项级

IPPROTO_IP这一级的套接字选项与IP协议存在密切联系，比如可用它们 IP头内的特定字段，以及向IP多播组增添一个套接字等等。许多这样的选项都声明于 Winsock.h和Winsock2.h这两个头文件内，而且采用了不同的值。要注意的是，假如装载的是 Winsock 1，那么必须包括正确的头文件，同时建立与Wsock32.lib函数库的链接关系。若装载的是Winsock 2，那么情况也是类似的，除了将 Winsock 2的头文件包括进来外，还要建立与 Ws2_32.lib的链接关系。在多播通信环境中，这一点尤其重要，因为两个版本的 Winsock均支持多播通信。除Windows CE的早期版本之外，其他所有Win32平台都提供了对多播通信的支持。而 Windows CE自2.1版之后，也开始提供了这方面的支持。

1. IP_OPTIONS

选项值类型	获取/设置	Winsock版本	说明
char[]	两者均可	1+	设置或获取IP头内的IP选项

通过该标志，可在IP头内设置各种IP选项。某些可能出现的选项包括：

- 安全和控制限制：RFC 1108。
- 记录路由：每个路由器都将自己的IP地址添加到头内（参见第13章的Ping程序示例）。

■ 时间标志（时间戳）：每个路由器都增添自己的IP地址及时间。
■ 松散源路由选择：为访问选项头内列出的每个 IP地址，便要求这个数据包。
■ 严格源路由选择：只有访问选项头内列出的那些 IP地址时，才要求用到这个数据包。

要注意的是，主机和路由器并不支持所有这些选项。

设置一个IP选项时，传至setsockopt调用内部的数据会遵照如图 9-2所示的数据结构。 IP选项头最长可达40字节。

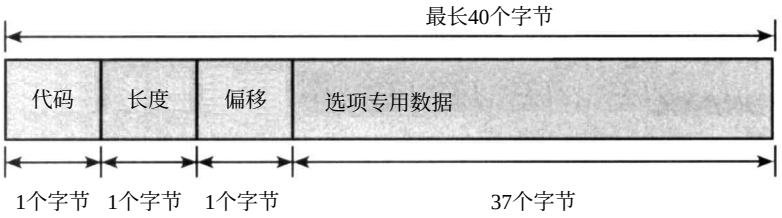

图9-2 IP选项头格式

其中，“ 代码 ”字段指出要提供的是什么类型的 IP选项。例如，若将该值设为 0x7，便意味着是“ 记录路由 ”选项。而“ 长度 ”字段对应着选项头的长度，“ 偏移 ”是指头内数据部分的起始偏移位置。头的“ 数据 ”部分必然是由特定的选项来决定的。在下述代码片断中，我们设置的是一个“ 记录路由 ”选项。请注意，我们首先声明了一个结构（struct ip_option_hdr），用它包含头三个选项值（代码、长度和偏移）。随后，我们将由具体选项决定的数据声明成一个数组，由9个无符号（无正负号）的长整数（ long）构成，因为要记录的数据最多允许包含9个IP地址。在此要注意的是， IP选项头允许的最大长度是 40字节；然而，我们的结构只占用了39字节。系统会自动对这个头进行填充，将其长度保持为一个 32位字的整数倍（最多 40字节）。

```
struct ip_option_hdr
{
    unsigned char    code;
    unsigned char    length;
    unsigned char    offset;
    unsigned long    addrs[9];
} opthdr;

...

ZeroMemory((char *)&opthdr, sizeof(opthdr));
opthdr.code = 0x7;
opthdr.length = 39;
opthdr.offset = 4;  // Offset to first address (addrs)
ret = setsockopt(s, IPPROTO_IP, IP_OPTIONS, (char *)&opthdr,
        sizeof(opthdr));
```

选项设好之后，它便可应用于在指定套接字上发出的所有数据包。以后，我们可随IP_OPTIONS一道，调用getsockopt函数，以取得已设置了的选项。然而，这样做并不会返回填充到选项专用缓冲区内的任何数据。为取得在 IP选项中设置的数据，要么必须将套接字创建成一个“ 原始套接字 ”（SOCK_RAW），要么应该设置IP_HDRINCL选项。在后一种情况下，IP头会在经过对一个Winsock获取函数的调用之后，随数据一道返回。

2. IP_HDRINCL

选项值类型	获取/设置	Winsock版本	说明
布尔值	两者均可	2+	如果是TRUE，IP头就会随即将发送的数据一起提交，并从读取的数据中返回

若将IP_HDRINCL选项设为TRUE，发送函数会将IP头包括在自己发送的数据前面，并会促使接收函数将IP头包括为自己的数据的一部分。因此，在我们调用一个 Winsock发送函数的时候，必须在数据前面包括完整的 IP头，并对IP头的每个字段进行正确的填写。在图 9-3中，我们向大家展示了IP头实际看起来的样子。该选项仅适用于 Windows 2000操作系统。

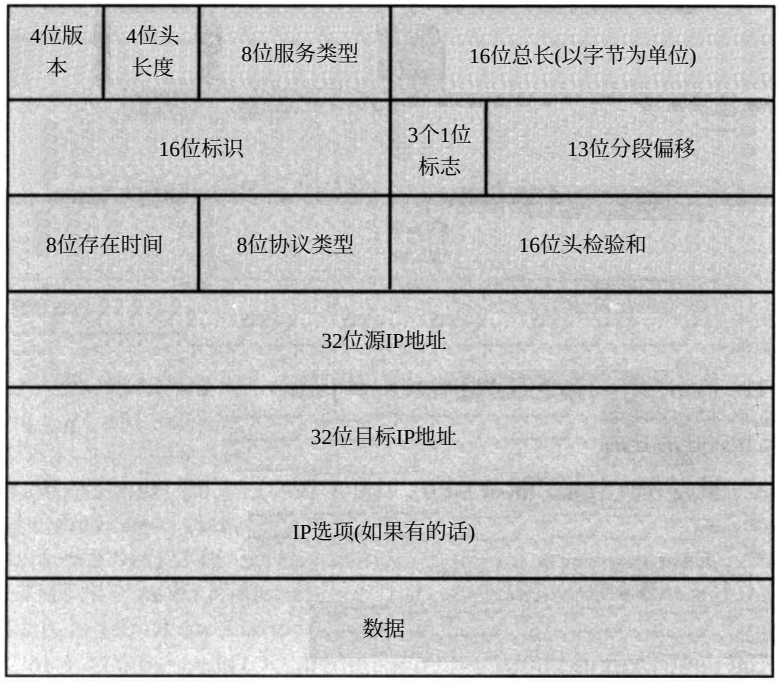

图9-3 IP头

其中，头的第一个字段指定的是 IP版本，目前通常是版本 4。头长度是指在整个头中， 32位字一共有多少个（一头的长度必须是 32位的整数倍）。接下来的字段是“ 服务类型 ”（TOS）。对此，大家可参考下文介绍的 IP_TOS套接字选项，了解进一步的情况。总长字段是指 IP头和数据加在一起，一共有多长（以字节计）。标识字段是一个独一无二的值，用于对发出的每个IP包进行“ 唯一性 ”的标定。通常，每发出一个数据包，系统都会递增这个值。标志和分段偏移字段仅在IP包需要分割为较小的包时才会用到。而“ 存在时间 ”（Time to Live，TTL）字段，则用于限制数据包最多可穿越多少个路由器。每遇到一个路由器对这个包进行转发的时候，TTL的值都会递减1。一旦TTL变成0，便会将这个包丢弃。这样一来，便可限制一个包能在网络上“ 生存 ”的时间，避免它无休止地“ 游荡 ”下去。协议字段用于对进入的数据包组装。采用了IP定址机制的某些有效协议包括 TCP，UDP，IGMP和ICMP等等。校验和是指对整个IP头进行16位1的求余总和结果。要注意的是，这种计算仅针对头进行，不针对实际的数据。接下来的两个字段分别是 32位的IP源和目标地址。 IP选项字段是一个长度不定的字段，包含了某些可选的信息，通常与 IP安全或路由选择有关。

要想将一个IP头和打算发送的数据组装到一起，最简单的办法便是定义一个结构，在其

中包含IP头以及数据，再将整个结构传递给 Winsock的send调用。大家可参考第 13章的内容，那里除详细讲述了这方面的知识之外，还提供了与这个选项对应的一个例子。要注意的是，该选项仅适用于Windows 2000。

3. IP_TOS

选项值类型	获取/设置	Winsock版本	说明
整数	两者均可	1+	IP服务类型

服务类型（TOS）是在IP头中设置的一个字段，用于指出与一个包对应的具体特征。该字段总长度为8位，总共划分为三部分：一个 3位长度的优先级字段（这方面的设置会被忽略），一个4位长度的TOS字段，以及一个补足位（必须设为 0）。其中，四个TOS位分别对应于最短延迟、最高吞吐速度、最大可靠性以及最小费用。但要注意的是，每一次只能设置一个位。若这四个位的值均为0，便暗示着采用“普通服务”。在RFC 1340文件中，针对各种标准应用（包括TCP、SMTP、NNTP等等），分别推荐了不同的设置。此外， RFC 1349文件也对早先公布的RFC文件进行了一些补充及纠正。

对某些交互式应用而言，比如 Rlogin或Telnet，它们可能想将延迟时间缩至最短。而对任何一种文件传输机制来说（如 FTP），都希望将重点放在提高数据的吞吐速度上。至于“最大可靠性”，则是网络管理（ SNMP）和路由协议所强烈要求的。最后， Usenet新闻协议（NNTP）是需要将金钱方面的开销降至最低程度的一个典型例子。注意 IP_TOS选项并不适用于Windows CE。

在提供服务质量（QoS）的套接字上尝试设置TOS位时，还有另一个问题需要注意。由于IP优先级设定已被QoS利用，以便对不同的服务等级加以区分，所以并不推荐开发者任意更改这些值。因此，若在具有QoS功能的套接字上调用setsockopt函数，同时又设置了IP_TOS选项，那么QoS服务提供者会事先拦截这一调用，查验是否对此项设定进行了修改。大家可参阅第12章，了解QoS的详情。

4. IP_TTL

选项值类型	获取/设置	Winsock版本	说明
整数	两者均可	1+	IP协议的“存在时间”（TTL）参数

“存在时间”（TTL）字段需要在IP头内进行设置。数据报利用 TTL字段对数据报能够通过的最大路由器数量加以限制。之所以要进行这样的限制，目的在于避免由于路由循环，造成数据报永无休止地在网络中“游荡”。其背后的道理在于，数据报每经过一个路由器，都会由路由器解析出它的 TTL值，并将其减去 1。若发现这个值变成了 0，数据报便会被“无情”地丢弃。但要注意的是， Windows CE不支持这个选项。

5. IP_MULTICAST_IF

选项值类型	获取/设置	Winsock版本	说明
无符号的长整数	两者均可	1+	获取或设置打算从它上面发出多播数据的本地接口

IP多播接口（IF）选项用于设置一个本地接口，本地机器以后发出的任何多播数据都会经由它传送出去。只有在那些同时安装了多个网络接口（网卡、 Modem等等）的机器上，此项设置才有意义。optval参数应当是一个无符号（无正负号）的长整数值，对应于本地接口的二进制IP地址。可用inet_addr函数将一个字串形式的IP地址（点分十进制）转换成一个无符号的长整数值，如下例所示：

```
DWORD    mcastIF;
```

```
// First join socket s to a multicast group
mcastIF = inet_addr("129.113.43.120");
ret = setsockopt(s, IPPROTO_IP, IP_MULTICAST_IF, (char *)&mcastIF,
    sizeof(mcastIF));
```

6. IP_MULTICAST_TTL

选项值类型	获取/设置	Winsock版本	说明
整数	两者均可	1+	为套接字获取或设置多播数据包的存在时间

和IP的TTL设置类似，这个选项执行相同的功能，只是它仅适用于通过指定套接字发出的多播数据。同样地，TTL的目的是防止路由循环。但在多播通信环境中，设置 TTL可缩窄数据的“传播”范围。因此，多播组的各个成员必须在一个有效的“范围”之内，才能接收到数据报。多播数据报采用的默认 TTL设定是1。

7. IP_MULTICAST_LOOP

选项值类型	获取/设置	Winsock版本	说明
布尔值	两者均可	1+	如果是TRUE，发至多播地址的数据将原封不动地“反射”或“反弹”回套接字的进入缓冲区

默认情况下，在我们发送IP多播数据的时候，假如套接字亦属于那个多播组的一名成员，数据便会原封不动地回返至套接字。若将该选项设为 FALSE，发出的任何数据都不会投递至套接字的进入数据队列中。

8. IP_ADD_MEMBERSHIP

选项值类型	获取/设置	Winsock版本	说明
struc ip_mreq	只能设置	1+	在指定的IP组内为套接字赋予成员资格

该选项实际是由Winsock 1提供的一个方法，可将套接字加入一个指定的 IP多播组。此时，需要用 socket函数来创建地址家族为 AF_INET的一个套接字，同时将套接字的类型设为SOCK_DGRAM。要想将套接字加入一个多播组，请使用下述结构：

```
struct ip_mreq
{
    struct in_addr  imr_multiaddr;
    struct in_addr  imr_interface;
};
```

在ip_mreq结构中， imr_multiaddr对应于打算加入的那个多播组的二进制格式地址；而imr_interface对应于打算用来收发多播数据的那个本地接口。大家可参考第 11章，对哪些多播地址才算“有效”，做到心中有数。imr_interface字段要么设为本地接口的一个二进制 IP地址，要么设为INADDR_ANY，表明选择的是默认接口。

9. IP_DROP_MEMBERSHIP

选项值类型	获取/设置	Winsock版本	说明
struct ip_mre1	只能设置	1+	将套接字从指定的IP组内删去（撤消成员资格）

该选项正好与IP_ADD_MEMBERSHIP相反。随ip_mreq结构调用该选项，并在结构中放入与当初加入指定多播组时相同的值，套接字 s便会从那个组内删去，脱离成员关系。同样地，第11章详细讲述了IP多播的知识。

10. IP_DONTFRAGMENT

选项值类型	获取/设置	Winsock版本	说明
布尔值	两者均可	1+	如果是TRUE，就不对IP数据报进行分段

这个标志指示网络在传输过程中，不要对 IP数据报进行分段处理。然而，假如一个 IP数据

报的大小已经超过了最大传输单元（MTU）值，但未在IP头内设置分段标志，这个数据报就会被丢弃，同时向发送者返回一条ICMP错误消息（“需要分段，但未设置分段位”）。要注意的是，该选项并不适用于Windows CE。

9.1.5 IPPROTO_TCP选项级别

仅有一个选项从属于IPPROTO_TCP级别。该选项只适用于流式套接字（SOCK_STREAM），其地址家族为AF_INET。这个选项可用在所有Winsock版本上，并得到了所有Win32平台的支持，包括Windows CE。

TCP_NODELAY

选项值类型	获取/设置	Winsock版本	说明
布尔值	两者均可	1+	若为TRUE，就会在套接字上禁用Nagle算法

为减少网络通信的开销，提升性能以及吞吐速度，系统默认采用Nagle算法。若应用程序请求发送一批数据（量较大），那么系统在接收了那些数据之后，可能会稍候一段时间，等其他数据累积进来，最后统一发送出去。但假如在一段规定的时间内，并无新数据加入，那么原先那些数据当然也会不管三七二十一地发送出去。这样造成的一个好结果便是：在单独一个TCP包内，数据量增大了。与之相反的则是：使用多个TCP包，但每个包携带的数据量比较少。如果是后一种情况，那么必然会涉及的一项“开销”在于，对每个包来说，其TCP头都必须占据20个字节的长度。例如，假定在此只发送2个字节，那么20个字节的头便显得有点儿多余。因此，在采用了Nagle算法后，可以更有效地利用数据包的可用空间。该算法的另一个功能是收到确认消息的延迟发送。系统收到TCP数据之后，必须向对方反馈回一条ACK（收到确认）消息。但采用了该算法后，主机会暂时等待一段时间，看看是否有需要发给对方的数据，以便能随那些数据一道，将ACK消息反馈回去，从而节省一个数据包的通信量（这又是一项开销）。

本选项的目的便是禁止采用Nagle算法，因为在某些情况下，它的行为反而会产生不利的影响。若网络应用通常只需发送数量相当少的数据，同时要求能得到极其迅速的响应，那么再使用这种算法，反而会影响性能。Telnet便是这样的一个典型例子。Telnet的本质是一种“交互式”或“互动式”应用，用户可通过它登录至一台远程机器，然后向其传送命令。通常，用户每秒钟只会进行少量的键击。若Nagle在此仍要不知好歹地“发挥作用”，便会造成响应的迟钝，甚至造成对方主机不予应答的错觉。

9.1.6 NSPROTO_IPX选项级别

这儿列出的套接字选项均属Microsoft公司对Windows IPX/SPX套接字接口作出的特有扩展，用于确保与现有应用的兼容性。若非考虑到兼容性方面的原因，我们并不建议使用这些选项，因为它们仅适用于Microsoft IPX/SPX堆栈。若某个应用程序利用了这些扩展，那么完全有可能无法在其他IPX/SPX系统中正常工作。这些选项均定义在WSNwLink.h头文件中。但在程序中定义时，应将该文件包括于Winsock.h及Wsipx.h这两个文件的后面。

1. IPX_PTYPE

选项值类型	获取/设置	Winsock版本	说明
整数	两者均可	1+	获取或设置IPX包的类型

该选项用于设置或获取IPX数据包的类型。在自这个套接字发出的每个IPX包内，optval

参数中指定的值都对应于数据包的类型。optval参数指定的是一个整数值。

2. IPX_FILTERPTYPE

选项值类型	获取/设置	Winsock版本	说明
整数	两者均可	1+	获取或设置准备过滤的IPX包之类型

该选项用于获取或设置接收过滤器数据包类型。注意在任何接收调用中，只有类型与optval参数中指定的值相符的IPX包才会返回；若类型与指定的数据包类型不符，数据包便会被丢弃。

3. IPX_STOPFILTERPTYPE

选项值类型	获取/设置	Winsock版本	说明
整数	只能设置	1+	删除为指定IPX包设置的过滤器

可用该选项停止过滤当初用IPX_FILTERTYPE选项设置的数据包类型。

4. IPX_DSTYPE

选项值类型	获取/设置	Winsock版本	说明
整数	两者均可	1+	获取或设置SPX头中的数据流字段值

针对发出的每个数据包的SPX头，该选项用于获取或设置数据流字段的值。

5. IPX_EXTENDED_ADDRESS

选项值类型	获取/设置	Winsock版本	说明
布尔值	两者均可	1+	如果是TRUE，便允许对IPX包进行扩展定址

该选项用于允许或禁止扩展定址／寻址。发送数据时，它会在SOCKADDR_IPX结构中加入unsigned char sa_ptype，使结构的总长变成15字节；而在接收数据时，该选项会在SOCKADDR_IPX结构中同时添加sa_ptype和unsigned char sa_flags元素，使其总长变成16字节。目前在sa_flags中定义的位是：

- 0x01：接收的帧是以广播形式发送出去的。
- 0x02：接收的帧是自这台机器发送出去的。

6. IPX_RECVHDR

选项值类型	获取/设置	Winsock版本	说明
布尔值	两者均可	1+	如果是TRUE，就随接收调用一起，返回IPX头

若将该选项设为“真”，那么任何Winsock接收调用都会随正式数据一道，返回IPX头。

7. IPX_MAXSIZE

选项值类型	获取/设置	Winsock版本	说明
整数	只能获取	1+	返回IPX数据报的最大长度

随该选项调用getsockopt函数，便可返回允许的最大IPX数据报长度。

8. IPX_ADDRESS

选项值类型	获取/设置	Winsock版本	说明
IPX_ADDRESS_DATA	只能获取	1+	返回具备IPX能力之适配器的有关信息

该选项用于对IPX绑定的一个特定适配器进行查询。在同时安装了n个网络适配器（网卡、Modem等等）的系统中，适配器的编号范围在0到n~1之间。如欲调查系统内到底安装了多少个提供IPX能力的适配器，可在设置IPX_MAX_ADAPTER_NUM选项的前提下，调用getsockopt；或者重复调用IPX_ADDRESS，同时递增adapternum的值，直至返回错误。optval参数指向的是一个IPX_ADDRESS_DATA结构。下面是该结构的定义：

```
typedef struct _IPX_ADDRESS_DATA
{
```

```
    INT     adapternum;  // Input: 0-based adapter number
    UCHAR   netnum[4];   // Output: IPX network number
    UCHAR   nodenum[6];  // Output: IPX node address
    BOOLEAN wan;         // Output: TRUE = adapter is on a WAN link
    BOOLEAN status;      // Output: TRUE = WAN link is up (or adapter
                         // is not WAN)
    INT     maxpkt;      // Output: max packet size, not including IPX
                         // header
    ULONG   linkspeed;   // Output: link speed in 100 bytes/sec
                         // (i.e., 96 == 9600 bps)
} IPX_ADDRESS_DATA, *PIPX_ADDRESS_DATA;
```

9. IPX_GETNE TINFO

选项值类型	获取/设置	Winsock版本	说明
IPX_NETNUM_DATA	只能获取	1+	返回与一个指定IPX网络编号有关的信息

这个选项包含了与一个特定 IPX网络编号有关的信息。若网络刚好位于 IPX的缓冲之中，那么选项会直接将信息返回；否则的话，它会发出 RIP请求，以期找到目标网络。 optval参数指向的是一个有效的IPX_NETNUM_DATA结构。对这个结构的定义如下：

```
typedef struct _IPX_NETNUM_DATA
{
    UCHAR  netnum[4];  // Input: IPX network number
    USHORT hopcount;   // Output: hop count to this network, in machine
                       // order
    USHORT netdelay;   // Output: tick count to this network, in machine
                       // order
    INT    cardnum;    // Output: 0-based adapter number used to route
                       // to this net; can be used as adapternum input
                       // to IPX_ADDRESS
    UCHAR  router[6];  // Output: MAC address of the next hop router,
                       // zeroed if the network is directly attached
} IPX_NETNUM_DATA, *PIPX_NETNUM_DATA;
```

10. IPX_GETNETINFO_NORIP

选项值类型	获取/设置	Winsock版本	说明
IPX_ADDRESS_DATA	两者均可	1+	如果是TRUE，就不会对IP数据报进行分段

该选项类似于IPX_GETNETINFO，只是它并不发出RIP请求。若网络正好在IPX的缓冲之内，信息便可直接返回；否则，调用就会失败，不会发出更多的请求。大家同时可参考IPX_RERIPNETNUMBER，后者无论如何都会发出RIP请求。和IPX_GETNETINFO相似，这个选项要求以optval参数的形式，传递一个IPX_NETNUM_DATA结构。

11. IPX_SPXGETCONNECTIONSTATUS

选项值类型	获取/设置	Winsock版本	说明
IPX_SPXCONNSTATUS_DATA	只能获取	1+	返回与一个已建立连接的SPX套接字有关的信息

该选项可返回与一个已建立连接的 SPX套接字有关的信息。 optval参数指向的是一个IPX_SPXCONNSTATUS_DATA结构，定义见下。所有数字都按网络字节顺序排列（从高到低）。

```
typedef struct _IPX_SPXCONNSTATUS_DATA
{
    UCHAR  ConnectionState;
    UCHAR  WatchDogActive;
    USHORT LocalConnectionId;
```

```
    USHORT RemoteConnectionId;
    USHORT LocalSequenceNumber;
    USHORT LocalAckNumber;
    USHORT LocalAllocNumber;
    USHORT RemoteAckNumber;
    USHORT RemoteAllocNumber;
    USHORT LocalSocket;
    UCHAR  ImmediateAddress[6];
    UCHAR  RemoteNetwork[4];
    UCHAR  RemoteNode[6];
    USHORT RemoteSocket;
    USHORT RetransmissionCount;
    USHORT EstimatedRoundTripDelay; /* In milliseconds */
    USHORT RetransmittedPackets;
    USHORT SuppressedPacket;
} IPX_SPXCONNSTATUS_DATA, *PIPX_SPXCONNSTATUS_DATA;
```

12. IPX_ADDRESS_NOTIFY

选项值类型	获取/设置	Winsock版本	说明
IPX_ADDRESS_DATA	只能获取	1+	若IPX适配器的状态发生改变，则发出异步通知

该选项可提交一个请求，以便在IPX与之绑定的那个适配器的状态发生更新后，及时收到通知。注意状态的改变通常是在WAN线路建立或断开的时候发生的。该选项要求调用者提交一个IPX_ADDRESS_DATA结构，将其放在 optval参数中传递。然而，一个例外是在IPX_ADDRESS_DATA后面，要紧随着一个句柄，对应于一个尚未进入传信状态（即尚未触发）的事件。下述伪代码向大家阐释了调用该选项的一个方法：

```
char  buff[sizeof(IPX_ADDRESS_DATA) + sizeof(HANDLE)];
IPX_ADDRESS_DATA *ipxdata;
HANDLE *hEvent;

ipxdata = (IPX_ADDRESS_DATA *)buff;
hEvent = (HANDLE *)(buff + sizeof(IPX_ADDRESS_DATA));
ipxdata->adapternum = 0;  // Set to the appropriate adapter
*hEvent = CreateEvent(NULL, TRUE, FALSE, NULL);
setsockopt(s, NSPROTO_IPX, IPX_ADDRESS_NOTIFY, (char *)buff,
sizeof(buff));
```

在提交了 getsockopt查询之后，它会成功完成任务。然而，由 optval指向的那个IPX_ADDRESS_DATA结构在那时却不会马上发生更新。相反，请求会在传送层内部进入队列（排队）；以后若适配器的状态发生了变化， IPX就会找到队列中的一个 getsockopt查询，并填好IPX_ADDRESS_DATA结构中的各个字段。随后，它会传送由 optval缓冲区内的句柄指向的那个事件。若同时提交多个 getsockopt调用，那么必须使用不同的事件。之所以要利用事件，是由于调用需要“ 异步 ”进行； getsockopt目前尚不支持这一功能。

警告 以目前的实施方案来看，针对状态发生的每一种变化，传送层都只会传信队列中的一个查询。因此，在同一个时候，只能运行使用了排队查询的一个服务。

13. IPX_MAX_ADAPTER_NUM

选项值类型	获取/设置	Winsock版本	说明
整数	只能获取	1+	返回存在的IPX适配器个数

该选项可返回系统中具有 IPX功能的适配器的数量。若该调用返回 n个适配器，那么适配

器的范围编号便在0到n~1之间。

14. IPX_RERIPNETNUMBER

选项值类型	获取/设置	Winsock版本	说明
IPX_NETNUM_DATA	只能获取	1+	返回一个网络编号的相关信息

该选项和IPX_GETNETINFO颇为类似，只是它会强迫IPX重发RIP请求，即便目标网络当前已位于它的缓冲区内（然而，假若它与目标网络直连，却不必重发）。和IPX_GETNETINFO类似，它也要求通过optval参数，传递一个IPX_NETNUM_DATA结构。

15. IPX_RECEIVE_BROADCAST

选项值类型	获取/设置	Winsock版本	说明
布尔值	只能设置	1+	如果是TRUE，就不接收IPX广播包

默认情况下，IPX套接字有能力接收广播数据包。若应用程序不需要收取广播包，则应将该选项设为 FALSE，从而在一定程度上改善系统性能。但要注意的是，即使将选项设为FALSE，也并不一定造成为那个应用过滤掉广播数据。

16. IPX_IMMEDIATESPXZCK

选项值类型	获取/设置	Winsock版本	说明
布尔值	两者均可	1+	如果是TRUE，就不在SPX连接上延迟发送ACK

如将该选项设为TRUE，那么确认包（ACK）不会在SPX连接之上推迟发送。若应用程序不需要在SPX通信链路上进行双向通信，便可尽情地将其设为 TRUE。这样做尽管会增加发送的ACK包的数量，但却能防止应用程序由于推迟发送确认消息，造成对性能的影响。

9.2 IOCTLSOCKET和WSAIOCTL

一系列套接字I/O控制函数用于在套接字之上，控制I/O的行为，同时获取与那个套接字上进行的I/O操作有关的信息。其中，第一个函数是 ioctlsocket，起源于Winsock 1规范，声明如下：

```
int ioctlsocket (
    SOCKET s,
    long cmd,
    u_long FAR *argp
);
```

其中，参数s指定的是要在上面采取I/O操作的套接字描述符，而cmd是一个预定义的标志，用于打算执行的I/O控制命令。最后一个参数 argp对应的是一个指针，指向与命令密切相关的一个变量。描述好每个命令之后，再给出要求变量的类型。 Winsock 2引入了一个新的ioctl函数，增添了数量多得多的新选项。首先，它将单个 argp参数分解成了一系列输入参数，用于容纳传递到函数内部的值；同时提供一系列输出参数，用于容纳自调用返回的数据。此外，函数调用可使用重叠I/O。这个新函数便是WSAIoctl，它的定义如下：

```
int WSAIoctl(
    SOCKET s,
    DWORD dwIoControlCode,
    LPVOID lpvInBuffer,
    DWORD cbInBuffer,
    LPVOID lpvOutBuffer,
    DWORD cbOutBuffer,
    LPDWORD lpcbBytesReturned,
```

```
    LPWSAOVERLAPPED lpOverlapped,
    LPWSAOVERLAPPED_COMPLETION_ROUTINE lpCompletionRoutine
);
```

头两个参数与ioctlsocket的相同。另两个参数（lpvInBuffer和cbInBuffer）则对输入参数进行了描述。其中，lpvInBuffer参数是一个指针，指向传递进入的值，而cbInBuffer指定的是数据的多少，以字节为单位。类似地，lpvOutBuffer和cbOutBuffer用于自调用返回的任何数据。lpvOutBuffer参数指向的是一个数据缓冲区，其中放置了返回的所有信息。而cbOutBuffer参数对应的是在lpvOutBuffer中传递进来的缓冲区的字节长度。要注意的是，某些调用可能只使用了输入或输出参数，而另一些调用两类参数都会用到。第七个参数是lpcbBytesReturned，对应于实际返回的字节数。最后两个参数是lpOverlapped和lpCompletionRoutine，在随重叠I/O调用这个函数时使用。大家可参考第8章，了解重叠I/O的使用详情。

9.2.1 标准I/O控制命令

1. FIONBIO

函数	输入	输出	Winsock版本	说明
ioctlsocket/WSAIoctl	无符号	无	1+	将套接字置入非锁定模式

该命令可在套接字s上允许或禁止“非锁定”（Nonblocking）模式。默认情况下，所有套接字在创建好后，都会自动进入“锁定”套接字。若随FIONBIO这个I/O控制命令来调用ioctlsocket，那么应设置argp，令其传递指向一个“无符号”（无正负号）长整数的指针；若打算启用非锁定模式，应将那个长整数的值设为一个非零值。而若设为0值，意味着套接字进入锁定模式。如换用新的WSAIoctl命令，只需在lpvInBuffer参数中简单地传递那个无符号长整数即可。

调用WSAAsyncSelect或WSAEventSelect函数的时候，会将套接字自动设为非锁定模式。调用了其中任何一个函数之后，再有任何将套接字设回锁定模式的企图，都会以失败告终，并返回WSAEINVAL错误。要想将套接字改回锁定模式，应用程序首先必须禁止WSAAsyncSelect。具体的做法是调用WSAAsyncSelect，同时令其lEvent参数等于0。或者调用WSAEventSelect，令lNetworkEvents参数等于0，从而禁止WSAEventSelect。

2. FIONREAD

函数	输入	输出	Winsock版本	说明
两者均可	无	无符号长整数	1+	返回准备在套接字上读取的数据量

该命令用于决定可从套接字上自动读入的数据量。对ioctlsocket来说，argp值会返回一个无符号的整数，其中包含了打算读入的字节数。而若使用WSAIoctl，那么无符号的整数是在lpvOutBuffer中返回的。若套接字s是一个“面向数据流”的套接字（类型为SOCK_STREAM），那么FIONREAD会返回在单独一次接收调用中，可返回的数据总量。要注意的是，若使用这种或其他形式的消息预先“窥视”机制，并一定保证能够返回正确的数据量。若在一个数据报套接字（类型为SOCK_DGRAM）上使用I/O控制命令，返回值就是在套接字上排队的第一条消息的大小。

3. SIOCATMARK

函数	输入	输出	Winsock版本	说明
两者均可	无	布尔值	1+	判断是否已读取了带外数据

若一个套接字配置成接收带外（OOB）数据，而且已设置成以内嵌方式读取这种OOB数

据（通过设置SO_OOBINLINE套接字选项），那么本I/O控制命令就会返回一个布尔值，指出接下来是否准备接收OOB数据。如答案是肯定的，则返回TRUE；否则，便返回FALSE，而且下一次接收操作会返回OOB数据之前的所有或部分数据。对ioctlsocket来说，argp会返回一个指向布尔变量的指针；而对WSAIoctl来说，会在lpvOutBuffer中返回指向布尔变量的指针。要记住的是，接收调用绝对不会将OOB数据和普通数据搅和在同一个调用中。请参考第7章，那里讲述了OOB数据的详细情况。

9.2.2 其他I/O控制命令

除那些与SSL处理有关的之外，本小节列出的所有I/O控制命令都是Winsock 2特有的。前者只适用于Windows CE平台。如仔细观察Winsock 2的头结构，便会发现其中实际还声明了另一些I/O控制命令。然而，本节只准备列出与用户的应用程序有关的那一部分I/O控制命令。此外，就象大家马上会看到的那样，并不是所有I/O控制命令都能在所有Win32平台上通用。相反，随着操作系统的升级换代，这些命令也必然会发生变化。对Winsock 2来说，其中大多数命令都是在Winsock2.h文件中定义的。而一些更新的命令（Windows 2000专用），则在Mstcpip.h中定义。

1. SIO_ENABLE_CIRCULAR-QUEUEING

函数	输入	输出	Winsock版本	说明
WSAIoctl	布尔值	布尔值	2+	如接收缓冲区队列溢出，则首先丢弃最早收到的消息

这个I/O控制命令在缓冲队列排满之后，用于控制基层服务提供者如何对进入的数据报消息加以控制。默认情况下，若负责存放进入数据的缓冲区满了，那么以后新收到的任何消息都会被丢弃。若将该选项设为TRUE，便表明新收到的消息绝对不会由于缓冲区的溢出而被丢弃；相反，队列中的那些“老”消息（最先收到的消息）会被丢弃，以便会新到的消息腾出地方。该命令仅适用于那些同不可靠的、面向消息的协议关联在一起的套接字。倘若在别的什么类型的套接字上使用这个I/O控制命令（比如采用一种面向数据流的协议的套接字），或者服务提供者本身不支持该命令，就会返回错误WSAENOPROTOOPT。要注意的是，该选项目前只能在Windows NT和Windows 2000平台上使用。

该I/O控制命令既可用于打开／关闭循环队列，亦可用于查询选项的最新状态。设置该选项时，只需使用输入参数。若对选项的最新状态进行查询，那么只需提供输出布尔参数。

2. SIO_FIND_ROUTE

函数	输入	输出	Winsock版本	说明
WSAIoctl	SOCKADDR	布尔值	2+	验证到指定地址的路由是否存在

该I/O控制命令用于核查是否能通过网络，访问到一个特定的地址。lpvInBuffer参数指向与指定协议对应的一个SOCKADDR结构。若目标地址已存在于本地缓冲，首先便会使其无效。对IPX来说，该调用会引发一个IPX GetLocalTarget调用，以便在网络中查询一个指定的远程地址。但不幸的是，目前各个Win32平台上的Microsoft提供者都还没有实现这个I/O控制命令。

3. SIO_FLUSH

函数	输入	输出	Winsock版本	说明
WSAIoct1	无	无	2+	判断是否已读取OOB数据

这个I/O控制命令可丢弃当前与指定套接字对应的发送队列的内容。该选项没有输入或输出参数。目前仅Windows 2000和Windows NT 4 Service Pack 4（SP4）才提供了对它的支持。

4. SIO_BROADCAST_ADDRESS

函数	输入	输出	Winsock版本	说明
WSAIoctl	无	SOCKADDR	2+	为套接字地址家族返回一个广播地址

这个I/O控制命令可返回一个SOCKADDR结构（通过lpvOutBuffer），其中包含了套接字s之地址家庭的广播地址，可在sendto或WSASendTo中使用。这个命令仅适用于Windows NT和Windows 2000操作系统。Windows 98及Windows 98均会返回WSAEINVAL错误。

5. SIO_GET_EXTENSION_FUNCTION_POINTER

函数	输入	输出	Winsock版本	说明
WSAIoctl	GUID	函数指针？	2+	取得基层提供者特有一个函数指针

这个I/O控制命令用于获取具体提供者特有的函数（并不属于Winsock标准规范的一部分）。若选定了一种提供者，程序员便可通过该I/O控制命令来利用其特有的函数，具体的做法是为每个函数都分配一个GUID。随后，应用程序可使用I/O控制命令SIO_GET_EXTENSION_FUNCTION_POINTER，取得指向该函数的一个指针。在头文件Mswsock.h中，定义了由微软自己增添的一系列Winsock函数，包括各自对应的GUID。例如，要查询目前安装的Winsock提供者是否支持TransmitFile函数，可用它的GUID对提供者进行查询。这个GUID是用下述代码定义的：

```
#define WSAID_TRANSMITFILE \
   {0xb5367df0,0xcbac,0x11cf,{0x95,0xca,0x00,0x80,0x5f,0x48,0xa1,0x92}}
```

取得了像TransmitFile这样的一个扩展函数的指针后，便可直接调用它，不必事先将自己的应用程序同Mswsock.lib库文件链接起来。这样实际可取消在Mswsock.lib中进行的一次中间函数调用。

大家可参阅Mswsock.h文件，了解还有哪些微软特有的扩展已定义了自己的GUID（微软也是一个“提供者”）。我们开发一个分层的服务提供者时，I/O控制命令是至关重要的一个环节。请参阅第14章，了解有关服务提供者接口的详情。

6. SIO_CHK_QOS

函数	输入	输出	Winsock版本	说明
WSAIoctl	DWORD	DWORD	2+	为指定的套接字设置QOS属性

这个I/O控制命令可用来检查QoS内部的六种状态，目前只有Windows 2000才提供了对它的支持。以下六个标志分别对应于六种不同的状态：

- ALLOWED_TO_SEND_DATA（允许发送数据）。
- ABLE_TO_RECV_RSVP（能够接收RSVP）。
- LINE_RATE（线路速率）。
- LOCAL_TRAFFIC_CONTROL（本地通信控制）。
- LOCAL_QOSABILITY（本地质量保证）。
- END_TO_END_QOSABILITY（端到端质量保证）。

其中，第一个标志ALLOWED_TO_SEND_DATA用在QoS等级建立好之后（使用SIO_SET_QOS建立），但又在接收到任何RSVP预约请求（RESV）消息之前。若收到一条RESV消息，意味着要求的带宽已分配给我们的数据流。在收到RESV消息之前，与套接字对应的数据流只会提供“尽最大努力”（Best-effort）服务。在本书第12章，还会就RSVP协议以及网络资源的预约进行详尽、深入的探讨。利用ALLOWED_TO_SEND_DATA标志，可检视

当前的“尽最大努力”服务是否足够保证由 SIO_SET_QOS请求的QoS服务等级。返回值要么是1——意味着当前的“尽最大努力”带宽已经足够使用；要么是 0——意味着目前的带宽尚不足以保证请求的服务等级。若 ALLOWED_TO_SEND_DATA标志的返回值是0，那么作为发送方的应用程序，应在发出数据之前，等候一条 RESV消息的抵达。

第二个标志是ABLE_TO_RECV_RSVP, 指出主机是否能在指定套接字绑定的那个接口上，接收与处理RSVP消息。返回值是1或0，分别指出能还是不能接收RSVP消息。

下一个标志是LINE_RATE，可返回“尽最大努力”能使线路通信速率达到多少，单位是每秒多少千位（kbit/s或Kbps）。若线路的通信速率未知，便返回一个 INFO_NOT_AVAIABLE（信息不可用）值。

最后三个标志指出在本地机器或网络上，是否提供了特定的功能。对所有这三个选项来说，若返回值是 1，便表明对应的功能是支持的；若返回 0，则表明不支持。另外，若返回 INFO_NOT_AVAIABLE，表明没有办法检查。LOACAL_TRAFFIC_CONTROL用于判断机器上是否安装了通信控制组件，而且是否能够使用。 LOCAL_QOSABILITY判断本地机器是否支持QoS。最后，END_TO_END_QOSABILITY指出整个本地网络是否具有 QoS能力。

7. SIO_GET_QOS

函数	输入	输出	Winsock版本	说明
WSAIoctl	无	QOS	2+	返回与套接字关联在一起的 QOS结构

这个I/O控制命令负责接收同套接字关联在一起的 QoS结构。提供的缓冲区必须足够大，以便容下整个结构。它有时会比 sizeof(QOS)稍大一些，因为结构中可能包含了提供者特有的一些信息。欲了解QoS的详情，可参考第12章。假如一个套接字的地址家族不支持 QoS，那么一旦在它上面使用该 I/O控制命令，便会返回 WSAENOPROTOOPT错误。该选项和 SIO_SET_QOS都只能在提供了 QoS相容传送协议的操作系统平台上使用，如 Windows 98和 Windows 2000。

8. SIO_SET_QOS

函数	输入	输出	Winsock版本	说明
WSAIoctl	QOS	无	2+	为指定套接字设置QOS属性

这个I/O控制命令与SIO_GET_QOS对应。该调用的输入参数是一个 QoS结构，定义了针对该套接字提出的带宽要求。这个调用不会返回任何输出值。大家可参考第 12章，了解有关 QoS的详情。要注意的是，这个选项和 SIO_GET_QOS都只适用于那些提供了 QoS相容传送协议的平台，如Windows 98和Windows 2000。

9. SIO_MULTIPOINT_LOOPBACK

函数	输入	输出	Winsock版本	说明
WSAIoctl	布尔值	布尔值	2+	设置或调查多播数据是否循环返回套接字

发送多播数据时，采取的默认行为是让发给多播组的任何数据都成为套接字接收队列内的进入数据。当然，只有在该套接字亦属于目标多播组的一名成员时，这种“循环返回”行为才有意义。目前，这种行为只能在 IP多播通信中见到，而 ATM多播并不支持。要想禁止循环返回特性，可在输入参数 lpvInBuffer中，传递一个布尔变量 FALSE。要想获取该选项当前的值，可将输入值留为NULL，并提供一个布尔变量，作为输出参数使用。

10. SIO_MULTICAST_SCOPE

函数	输入	输出	Winsock版本	说明
WSAIoctl	整数	整数	2+	设置或获取多播数据的存在时间值

这个命令控制着多播数据的“存在时间”，或者“作用域”。所谓“作用域”，实际是指数据最多允许路由至多少个网段。默认情况下，这个值的设定是 1。若多播包抵达一个路由器，其TTL值便会减1。一旦TTL值变成了0，那个包便会被“无情”地丢弃。要想对这个值进行设定，可在lpvInBuffer中，传递一个希望的TTL值（整数）；否则的话，可在调用 WSAIoctl的同时，令lpvOutBuffer指向一个整数，简单地取得当前的TTL设定。

11. SIO_KEEPLIVE_VALS

函数	输入	输出	Winsock版本	说明
WSAIoctl	tcp_keepalive	tcp_keepalive	2+	针对每一个连接，分别设置其TCP“保持活动”周期

这个I/O控制命令可针对每一个不同的连接，设置 TCP的“保持活动”消息发送周期。套接字选项SO_KEEPALIVE也允许发送TCP“保持活动”消息，但其发送间隔时间是在注册表（Registry）里设定的。若更改了注册表的值，同时也会更改机器上所有进程的“保持活动”周期。但若使用这个I/O控制命令，便可分别针对每一个套接字，设置其消息发送周期。要想在指定的一个套接字（已建立连接）上设置“保持活动”消息的发送周期，可初始化一个tcp_keepalive结构，并将其作为输入缓冲区传递。下面是该结构的定义：

```
struct tcp_keepalive
{
    u_long    onoff;
    u_long    keepalivetime;
    u_long    keepaliveinterval;
}
```

其中，结构字段 keepalivetime和keepaliveinterval的含义和本章早些时候讨论SO_KEEPALIVE选项时介绍的注册表值是完全一样的。一旦设好一个“保持活动”周期，便可对当前的设置进行查询。方法是调用 WSAIoctl，同时设置SIO_KEEPALIVE_VALS这个I/O控制命令，并提供一个 tcp_keepalive结构，作为输出缓冲区使用。这个 I/O控制命令只能在Windows 2000上使用。

12. SIO_RCVALL

函数	输入	输出	Winsock版本	说明
WSAIoctl	无符号整数	无	2+	接收网络上的所有数据包

使用这个I/O控制命令，同时将值设为 TRUE，便允许指定的套接字接收网络上的所有 IP包。这要求将套接字句柄传递给 WSAIoctl，同时套接字的地址家族必须是 AF_INET，套接字的类型必须是SOCK_RAW，而协议必须是IPPROTO_IP。除此以外，套接字必须同一个明确的本地接口建立绑定关系。换言之，我们不可将其绑定到所谓的 INADDR_ANY。一旦绑定好套接字，同时设好I/O控制命令，便可调用recv或WSARecv，以便返回IP数据报。要注意的是，由于这些数据是“数据报”，所以必须提供一个足够大的缓冲区。由于 IP头的总长字段是一个16位的整数倍，所以理论上的最大长度为 65535字节。但在实际应用中，网络的最大传输单元（MTU）要小得多。若使用这个 I/O控制命令，要求在本地机器上以“管理员”的身份登录。此外，这个I/O命令仅适用于Windows 2000。在本书的配套光盘上，有一个名为 Rcvall.c的示范文件，向大家阐述了如何使用这个命令，以及如何使用另外两个 SIO_RCVALL命令。

13. SIO_RCVALL_MCAST

函数	输入	输出	Winsock版本	说明
WSAIoctl	无符号整数	无	2+	接收网络上的所有多播数据包

这个I/O控制命令类似于上面讲述的 SIO_RCVALL。SIO_RCVALL的使用规则也适用于这

里，只是传递给WSAIoctl的套接字应当由通过IPPROTO_IGMP指定的协议来创建。另一个区别是此时返回的只有多播数据，而非返回全部 IP数据包。换言之，只有发至目标地址范围224.0.0.0到239.255.255.255之间的IP包，才会返回。该I/O控制命令仅适用于Windows 2000。

14. SIO_RCVALL_IGMPMCAST

函数	输入	输出	Winsock版本	说明
WSAIoctl	无符号整数	无	2+	接收网络上的所有IGMP数据包

同样，该命令类似于 SIO_RCVALL，只是传递给 WSAIoctl的套接字应当由与IPPROTO_IGMP对应的协议创建。若设置这个选项，那么只会返回 IGMP包。大家可参考前面对SIO_RCVALL的介绍，了解如何来使用这个选项。要注意的是，该 I/O控制命令只能用于Windows 2000。

15. SIO_ROUTING_INTERFACE_QUERY

函数	输入	输出	Winsock版本	说明
两者均可	SOCKADDR	无	2+	判断是否已读取OOB数据

利用这个I/O控制命令，我们可找到用来向远程机器发送数据的那个本地接口的地址。远程机器的地址应当以SOCKADDR结构的形式来传递，并通过lpvInBuffer参数来传递这个结构。除此以外，这里应该通过lpvOubBuffer参数，提供一个足够大的缓冲区。在这个缓冲区内，将包含一个或多个SOCKADDR结构，对那些可供使用的接口进行描述。该命令既可用于单播端点，亦可用于多播端点，而且自该调用返回的接口可在后续的 bind调用中使用。

Windows 2000的即插即用特性是提供这样一个I/O控制命令的动机。用户可能自由地插拔PCMCIA网卡，从而影响应用程序能够使用的网络接口。为 Windows 2000量身定造的一个应用程序应充分考虑到这方面的因素，并相应地采取对策。

另外，程序也不能仅仅依赖自SIO_ROUTING_INTERFACE_QUERY返回的信息。这些信息并不能保证其长时期有效。要想应付这方面的情况，必须同时使用 SIO_ROUTING_INTERFACE_CHANGE这个I/O控制命令，它负责在接口发生变化之后，向应用程序发出通知。一旦收到这样的通知，请再次调用SIO_ROUTING_INTERFACE_QUERY，获取最新信息。

16. SIO_ROUTING_INTERFACE_CHANGE

函数	输入	输出	Winsock版本	说明
WSAIoctl	SOCKADDR	无	2+	与一个端点连接的接口发生改变后，发出通知

用这个I/O控制命令指出自己希望在本地路由接口发生任何变化后，都向自己发出通知，那个接口原本用于访问一个指定的远程地址。使用该命令时，需要在输入缓冲区中，设置一个SOCKADDR结构，对应于自己想查询的那个远程地址。若成功完成，不会有任何数据返回。但是，假如同那个路由连接的接口已发生了某种形式的变化，应用程序就会及时地收到通知。随后，程序可调用SIO_ROUTING_INTERFACE_QUERY，判断实际应使用哪一个接口。

有几种方式都可发出对这个命令的调用。假如套接字处在“锁定”模式，那么除非接口发生变化，否则WSAIoctl调用不会完成并返回。若套接字处在“非锁定”模式，便会返回WSAEWOULDBLOCK错误。随后，应用程序可通过WSAEventSelect或WSAAsyncSelect，等候“路由更改”这一事件的发生，同时在网络事件们掩码中设置 FD_ROUTING_INTERFACE_CHANGE标志。另外，重叠 I/O亦可用来进行这样的调用。使用这个方法，我们可在WSAOVERLAPPED结构中提供一个事件句柄，它会在路由改变后收到通知。

输入SOCKADDR结构中指定的地址可以是一个具体的地址，亦可使用通配地址：INADDR_ANY，表明发生任何路由更改，自己都想收到通知。要注意的是，由于路由信息相对来说比较稳定，不大会发生变化，所以作为提供者（比如微软提供者），提供了忽略来自应用程序输入缓冲区的信息的选项，只在接口发生变化后，才发出通知。因此，一种更好的做法或许是先注册收到关于任何变化的通知，然后简单地调用 SIO_ROUTING_INTERFACE_QUERY，看看发生的改变是否会影响到自己的应用程序。

17. SIO_ADDRESS_LIST_QUERY

函数	输入	输出	Winsock版本	说明
WSAIoctl	无	SOCKET_ADDRESS_LIST	2+	返回套接字绑定的一系列接口的列表

这个I/O控制命令用于取得与套接字协议家族相符的一系列本地传送地址的列表，应用程序可与之建立绑定关系。此时的输出缓冲区是一个 SOCKET_ADDRESS_LIST结构，定义如下：

```
typedef struct _SOCKET_ADDRESS_LIST
{
INT              iAddressCount;
SOCKET_ADDRESS Address[1];
} SOCKET_ADDRESS_LIST, FAR * LPSOCKET_ADDRESS_LIST;

typedef struct _SOCKET_ADDRESS
{
    LPSOCKADDR lpSockaddr;
    INT        iSockaddrLength;
} SOCKET_ADDRESS, *PSOCKET_ADDRESS, FAR * LPSOCKET_ADDRESS;
```

其中，iAddressCount字段用于返回列表中地址结构的数量，而 Address字段对应的是一个数组，包含了与协议家族相符的一系列地址。

在Win32即插即用环境中，有效地址的数量可能会发生动态变化。因此，应用程序不可仅仅依赖自这个I/O控制命令返回的信息，来进行相关的操作。要想将动态改变的因素考虑在内，应用程序首先应该调用 SIO_ADDRESS_LIST_QUERY，取得最新的接口信息，然后调用SIO_ADDRESS_LIST_CHANGE，表明自己想收到与未来的变化有关的通知。一旦地址列表发生改变（收到了通知），应用程序就应重新查询一遍。

假如提供的输出缓冲区不够大， WSAIoctl调用便会失败，并返回 WSAEFAULT错误。同时，lcbBytesReturned参数会指出到底多大的缓冲区才够。

18. SIO_ADDRESS_LIST_CHANGE

函数	输入	输出	Winsock版本	说明
WSAIoctl	无	无	2+	本地接口发生变化时，发出通知

通过该命令，一旦指定套接字之协议家族的本地传送地址列表发生变化（应用程序可与这些地址建立绑定关系），程序便会收到通知。若调用成功完成，在输出参数中，不会返回任何信息。

有几个办法可发出对该命令的调用。假如套接字处于锁定模式，那么除非接口真的发生了某种变化，否则 WSAIoctl调用是不会完成并返回的。若套接字处于非锁定模式，则返回WSAEWOULDBLOCK错误。随后，应用程序可利用 WSAEventSelect或者WSAAsyncSelect调用，同时在网络事件位掩码中设置 FD_ADDRESS_LIST_CHANGE标志，从而等候“ 路由更

改”事件的发生。另外，亦可通过重叠 I/O发出这样的调用。采用这个方法，我们要在WSAOVERLAPPED结构中提供一个事件句柄；一旦路由发生改变，那个句柄便会进入“传信”状态。

19. SIO_GET_INTERFACE_LIST

函数	输入	输出	Winsock版本	说明
WSAIoctl	无	INTERFACE_INFO[]	2+	返回本地接口列表

这个I/O控制命令定义于Ws2tcpip.h头文件内，用于返回与本地机器上每个接口有关的信息。在输入这一块，不需要设置任何东西；而在输出参数中，会返回由一系列INTERFACE_INFO结构构成的一个数组。该结构的定义如下：

```
typedef struct _INTERFACE_INFO
{
    u_long          iiFlags;            /* Interface flags   */
    sockaddr_gen    iiAddress;          /* Interface address */
    sockaddr_gen    iiBroadcastAddress; /* Broadcast address */
    sockaddr_gen    iiNetmask;          /* Network mask      */
} INTERFACE_INFO, FAR * LPINTERFACE_INFO;

#define IFF_UP              0x00000001 /* Interface is up                 */
#define IFF_BROADCAST       0x00000002 /* Broadcast is supported          */
#define IFF_LOOPBACK        0x00000004 /* This is loopback interface      */
#define IFF_POINTTOPOINT    0x00000008 /* This is point-to-point interface*/
#define IFF_MULTICAST       0x00000010 /* Multicast is supported          */

typedef union sockaddr_gen
{
    struct sockaddr Address;
    struct sockaddr_in  AddressIn;
    struct sockaddr_in6 AddressIn6;
} sockaddr_gen;
```

其中，iiFlags成员字段可返回一个标志掩码，指出接口是否可用（IFF_UP），以及支持广播通信（IFF_BROADCAST），还是支持多播通信（IFF_MULTICAST）。此外，它也会指出接口处在“循环返回”模式（IFF_LOOPBACK），还是处在点到点通信模式（IFF_POINTTOPOINT）。在其他三个字段中，则分别包含了接口的地址、广播地址以及相应的网络掩码。

9.2.3 安全套接字层的I/O控制命令

“安全套接字层”（Secure Socket Layer，SSL）命令目前只得到了Windows CE的支持。Windows 95/98、Windows NT以及Windows 2000均未包含具有SSL能力的提供者。除仅适用于Windows CE之外，支持这些选项的唯一Winsock版本只有Winsock 1。

1. SO_SSL_GET_CAPABILITIES

函数	输入	输出	说明
WSAIoctl	无	DWORD	返回Winsock安全提供者的功能

该命令负责接收一系列特殊的标志，指出 Windows套接字安全提供者都具有哪些方面的能力。输出缓冲区必须是指向一个 DWORD（双字）位字字段的一个指针。目前，仅定义了SO_CAP_CLIENT标志。

2. SO_SSL_GET_FLAGS

函数	输入	输出	说明
WSAIoctl	无	DWORD	返回与套接字对应的s信道特有标志

这个命令可返回同一个特定套接字关联在一起的、 s信道所特有的一系列标志。输出缓冲区必须是一个指针，指向一个 DWORD位字段。请参考下述的SO_SSL_SET_FLAGS，了解哪些才是有效标志。

3. SO_SSL_SET_FLAGS

函数	输入	输出	说明
WSAIoctl	DWORD	无	设置套接字s信道特有标志

这儿的输入缓冲区必须是一个指针，指向一个 DWORD位字段。目前，仅设置了SSL_FLAG_DEFER_HANDSHAKE（推迟联络）标志，允许应用程序在切换到密文之前，先进行明文数据的收发。若通过代理服务器通信，便必须设置这个标志位。

通常，Winsock安全提供者会在Winsock的connect函数执行“安全联络”操作。然而，假如发现设置了这个标志，联络便会推迟到应用程序执行了 SO_SSL_PERFORM_HANDSHAKE控制代码之后进行。完成联络后，这个标志会被重设。

4. SO_SSL_GET_PROTOCOLS

函数	输入	输出	说明
WSAIoctl	无	SSLPROTOCOLS	返回安全提供者支持的协议列表

该命令可取得一系列协议构成的列表，所有协议均是这个套接字目前所支持的。输出缓冲区必须是一个指针，指向一个SSLPROTOCOLS结构。下面是该结构的定义：

```
typedef struct _SSLPROTOCOL
{
    DWORD dwProtocol;
    DWORD dwVersion;
    DWORD dwFlags;
} SSLPROTOCOL, *LPSSLPROTOCOL;
typedef struct _SSLPROTOCOLS
{
    DWORD       dwCount;
    SSLPROTOCOL ProtocolList[1];
} SSLPROTOCOLS, FAR *LPSSLPROTOCOLS;
```

对dwProtocol字段来说，有效的协议包括 SSL_PROTOCOL_SSL2，SSL_PROTOCOL_SSL3和SSL_PROTOCOL_PCT1。

5. SO_SSL_SET_PROTOCOLS

函数	输入	输出	说明
WSAIoctl	SSLPROTOCOLS	无	设置基层提供者应当支持的一个协议列表

这个I/O控制命令指定一系列协议的列表，均是提供者准备在这个套接字上支持的。其中，输入缓冲区必须是一个指针，指向前述的 SSLPROTOCOLS结构。

6. SO_SSL_SET_VALIDATE_CERT_HOOK

函数	输入	输出	说明
WSAIoctl	SSLVALIDATECERTHOOK	无	为SSL身份凭据的接受设置校验函数

该I/O控制命令可设置一个指针，指向套接字的身份凭据验证挂钩。它用于指定一个回调函数，在自远程通信方收到一系列证书后，由 Windows套接字安全提供者加以调用。在此，输入缓冲区必须是一个指针，指向 SSLVALIDATECERTHOOK结构，如下所述：

```
typedef struct
```

```
{
    SSLVALIDATECERTFUNC     HookFunc;
    LPVOID                  pvArg;
} SSLVALIDATECERTHOOK, *PSSLVALIDATECERTHOOK;
```

其中，HookFunc字段是指向一个证书验证回调函数的指针； pvArg是指向应用程序特有数据的一个指针，可由应用程序用于任何目的。

7. SO_SSL_PERFORM_HANDSHAKE

函数	输入	输出	说明
WSAIoctl	无	无	在已建立连接的套接字上开始安全联络操作

这个I/O控制命令可在一个已连接的套接字上初始化安全联络序列；其中， SSL_FLAG_DEFER_HANDSHAKE标志已在连接前设置好了。数据缓冲区并非必需的，但SL_FLAG_DEFER_HANDSHAKE标志会被重设。

9.2.4 ATM I/O控制命令

本小节列出的I/O控制命令是ATM协议家族特有的。它们非常基本，主要涉及对 ATM设备数量的调查，以及取得本地接口的 ATM地址，要了解ATM地址机制的详情请参阅第6章。

1. SIO_GET_NUMBER_OF_ATM_DEVICES

函数	输入	输出	Winsock版本	说明
WSAIoctl	无	DWORD	2+	返回ATM适配器的数量

这个I/O控制命令可用一个DWORD（双字）填写由lpvOutBuffer指向的输出缓冲区，指出系统内总共包含了多少个 ATM设备。每个设备都有一个对应的、独一无二的 ID，范围在0~（该命令返回的数字-1）之间。

2. SIO_GET_ATM_ADDRESS

函数	输入	输出	Winsock版本	说明
WSAIoctl	DWORD	ATM_ADDRESS	2+	为指定设备返回ATM地址

该I/O控制命令可取得与指定设备关联在一起的本地 ATM地址。在输入缓冲区中，需要为这个命令指定类型为DWORD的一个设备ID；而在由lpvOutBuffer参数指向的那个输出缓冲区中，最后会填入一个 ATM_ADDRESS结构，其中包含了可由后续 bind调用使用的一个本地ATM地址。

3. SIO_ASSOCIATE_PVC

函数	输入	输出	Winsock版本	说明
WSAIoctl	ATM_PVC_PARAMS	无	2+	将套接字与一个永久虚拟回路关联起来

这个I/O控制命令可将套接字与一个永久虚拟回路（ Permanent Virtual Circuit，PVC）关联到一起。PVC在输入缓冲区中指定，采用 ATM_PVC_PARAMS结构的形式。套接字应从属于AF_ATM地址家族。自该函数成功返回之后，应用程序便能开始数据的收发，好象连接已实际地建好（其实是“虚拟”的）一样。

ATM_PVC_PARAMS结构的定义如下：

```
typedef struct
{
    ATM_CONNECTION_ID   PvcConnectionId;
    QOS                 PvcQos;
} ATM_PVC_PARAMS;
```

```
typedef struct
{
    DWORD  DeviceNumber;
    DWORD  VPI;
    DWORD  VCI;
} ATM_CONNECTION_ID;
```

4. SIO_GET_ATM_CONNECTION_ID

函数	输入	输出	Winsock版本	说明
两者均可	无	ATM_CONNECTION_ID	2+	判断是否已经读取OOB数据

这个I/O控制命令可获取同套接字关联在一起的 ATM连接ID。自该函数成功返回后，由lpvOutBuffer参数指向的那个输出缓冲区会填入一个 ATM_CONNECTION_ID结构，其中包含了设备编号以及对应的 VPI/VCI值，它们均是早先在 SIO_ASSOICATE_PVC的条目中定义好的。

9.3 小结

从表面看，数量如此众多的套接字选项及 I/O控制命令似乎会将人搞得晕头转向。但是，只需深入地探索，便会发现通过它们，应用程序的设计可得到极大的简化，可访问具体协议独有的许多特征。此外，还可利用它们对应用程序进行细致的调节。某些情况下，程序必须使用一个或多个套接字选项及 I/O控制命令，否则便无法继续工作，比如在 AppleTalk和IrDA（红外线）通信的前提下。当然，大多数时候，应用程序每次只需使用少数几个选项便足够了。另外要注意的是，套接字选项及 I/O控制命令最麻烦的一点在于，并非每个选项或 I/O控制命令都能适用于所有版本的 Windows平台。这样一来，一旦涉及跨平台的兼容问题，应用程序的设计便务必非常小心。