第10章　名字注册和解析

本章，我们将全面论述Winsock 2中引入的名字注册和解析模型，它们都是与协议无关的。由于现在已经废弃了Winsock 1中引入的名字注册和解析方法，所以我们将不再对它进行讨论。首先，介绍名字注册和解析的重要性及其用法的背景知识，然后步步深入现有的各种不同的名字注册模型，最后说明Winsock 2中用于解析名字的函数。另外，还谈谈如何注册自己的服务，以供他人查询。

10.1　背景知识

名字注册是一个过程，把一个用户好用的名和具体协议地址关联在一起。主机名及其 IP地址便是例证。人们发现要记住一个工作站的地址（比如 157.54.185.186）非常麻烦。所以他们宁愿把自己的机器命名为一个更容易记的地址，比如“ ajones1 ”。在IP中，一项名为“ 域名命名系统 ”（DNS）会把IP地址映射成相应的名字。我们将在下一节详细地讨论名字空间 。

人们不仅希望能够注册和解析主机名，还希望能够映射自己的 Winsock服务器地址，以便于在客户机打算和服务器连接时，可获得服务器的地址。比方说，你有一个服务器，它运行的机器地址为157.64.185.186，端口为5000。如果它只在那台机器上运行，就可以把这台服务器的地址硬编码到客户机应用程序中。但如果你需要一个更为动态的方法，即在若干台机器运行的服务器时，就要考虑采用一个容错的分布式应用程序。如果一个服务器崩溃或过于繁忙，另一个应用就开始接替它，为客户机提供服务。这种情况下，要找到服务器事实上在哪个地址运行，是非常令人头疼的。最理想的情况是用若干个地址来注册自己的服务器 —— 命名为“ 容错分布式服务器 ”。另外，大家也许还希望动态更新一个已注册的服务及其地址。这便是名字注册和解析的核心，而本章将着重讨论 Winsock提供的一些适用于分布式服务器注册和名字解析的设计。

10.2　名字空间模型

深入Winsock函数之前，需要为大家讲讲大多数协议附带的各种名字空间模型。名字空间提供了一种能力，用一个友好名把具体的协议及其定址属性关联在一起。最常见的名字空间是针对IP的DNS和Novell针对IPX开发的NetWare目录服务（NDS）。这些名字空间在组成和实施各不相同，但它们的有些属性特别有助于我们理解如何通过 Winsock注册和解析名字。

名字空间有三种类型：动态的、静态的和固定的。动态名字空间允许人们即时注册服务。另外，还意味着客户机可以在运行时对这个服务进行查看。一般说来，动态名字空间依赖于周期性地广播服务信息，表示该服务可继续使用。动态命名空间有：“ 服务声明协议 ”（SAP）（用于NetWare环境）和AppleTalk的“ 名字绑定协议 ”（NBP）名字空间。

这三类名字空间中，静态名字空间的灵活性最小。在静态名字空间内注册一个服务，需要在规定时间内进行手工注册。这意味着无法通过 Winsock用静态名字空间注册一个服务名，因为它只有一种解析法。 DNS是一个静态名字空间。举个例子来说，你可以用 DNS手工把IP

地址和主机名输入一个文件，DNS服务利用这个文件来处理解析请求。

固定名字空间和动态名字空间一样，允许即时注册服务。但和动态名字空间不同的是，固定名字空间把注册信息保留在固定的地方上，比如说磁盘上的一个文件中。只有在服务请求被删除时，固定名字空间才会把这项服务条目删除。它的优点在于灵活，不会连续不断地广播任何一种类型的有用信息。缺点就是如果一个服务行为不佳（或者说编得糟糕），该服务便在不通知名字空间提供者删除其服务条目的情况下，不知所终。从而导致客户机错误地认为该服务仍然可用。NDS是一个固定名字空间。

名字空间的列举

现在，大家已经知道名字空间的各种属性，但一台机器上可用哪些名字空间呢？我们来看看。多数预先定义的名字空间的声明都在Nspapi.h头文件中。每个名字空间都有一个分配所得的整数值。表10-1列出了一些比较常见的名字空间，它们已获支持，并可用于Win32平台。返回的名字空间由工作站上安装的协议决定。比方说，如果一个工作站上没有安装IPX/SPX，就不会返回NS_SAP名字空间。

表10-1 已获支持的名字空间

名字空间	值	说 明
NS_SAP	1	SAP名字空间；用于IPX网络
NS_NDS	2	NDS名字空间；也用于IPX网络
NS_DNS	11	DNS名字空间；多见于TCP/IP网络和互联网
ND_NTDS	32	Windows NT域名空间；运行于Windows 2000的与协议无关的命名空间

在一台机器上安装IPX/SPX时，只支持SAP名字空间查询。如果想注册自己的服务，还需要安装“SAP代理服务”。某些情况下，需要“NetWare的客户机服务”（Client Service of NetWare）把本地的IPX接口地址准确无误地显示出来。如果没有这个服务，本地地址全部以0的形式出现。另外还必须增加一个NDS客户机，以便利用NDS名字空间。所有这些协议和服务都可通过“控制面板”得以增添。

Winsock 2提供了一种方法，即如何通过程序获得一份列表，表上列出系统上所有可用的名字空间。这是通过调用WSAEnumNameSpaceProviders函数来完成的。该函数的定义如下：

```
INT WSAEnumNameSpaceProviders (
    LPDWORD lpdwBufferLength,
    LPWSANAMESPACE_INFO lpnspBuffer
);
```

第一个参数作为lpnspBuffer提交的缓冲区的长度，它是由多个WSANAMESPACE_INFO结构组成的一个大型数组。如果该函数是通过一个不充裕的缓冲区调用的，就会失败，把lpdwBufferLength设为所需要的最小长度，则会导致WSAGetLastError返回WSAEFAULT错误。这个函数将返回的WSANAMESPACE_INFO结构数的多少返回，或在错误之后出现SOCKET_ERROR。WSANAMESPACE_INFO结构描述指定机器上安装的一个独立名字空间。它的格式如下：

```
typedef struct _WSANAMESPACE_INFO {
GUID NSProviderId;
DWORD dwNameSpace;
BOOL fActive;
```

```
DWORD dwVersion;
LPTSTR lpszIdentifier;
} WSANAMESPACE_INFO, *PWSANAMESPACE_INFO,
LPWSANAMESPACE_INFO;
```

事实上，这个结构有两种格式——Unicode和ANSI。Winsock 2头文件针对具体需要，灵活地定义了为WSANAMESPACE_INFO的相应结构。实际应用中，所有结构和 Winsock 2注册和名字解析函数都有两个版本：Unicode和ANSI。该结构的第一个成员 NSProviderId，是一个通用的唯一识别符（GUID），它对这个特殊的名字空间进行描述。 dWNameSpace字段是这个名字空间的整数常量，比如 NS_DNS或NS_NAP。fActive成员是一个布尔值，若是真，就表明该名字空间可用，并准备发出请求；反之则表示提供者未激活，不能发出特别引用提供者的请求。dwVersion字段只对这个提供者的版本进行识别。最后， lpszIdentifier是该提供者的一个描述性的字串识别符。

10.3 服务的注册

下一步便是如何设置自己的服务，并使其有用，让网络上的其他机器都知道它。这就是利用名字空间提供者注册一个服务，这样一来，打算与之通信的客户机就可以对它进行声明或请求。注册一个服务实际上只有两步。第一步是安装一个描述服务特征的 service class（服务类）。

弄清楚服务类和事实上的服务本身之间的区别非常重要的。服务类和服务对两个不相同的概念。比方说，服务类描述哪些名字空间可用来注册自己的服务，该服务的特征有哪些（它是面向连接的，还是无连接的）。至于客户机如何才能建立一个连接，该服务类是无法将其描述出来的。服务类一旦注册，便可注册事实上的服务了，事实上的服务引用的是它所属的那个准确的服务类。一旦发生这种情况，客户机就可执行请求，在服务实例运行的地方进行查找，从而与别的机器通信。

10.3.1 安装服务类

在注册一个服务实例时，需要定义你的服务属于哪个服务类。用哪个名字空间注册这个服务类中的服务，由服务类定义决定。注册服务类的 Winsock函数是WSAInstallServiceClass，它的定义如下：

```
INT WSAInstallServiceClass (LPWSASERVICECLASSINFO lpServiceClassInfo);
```

唯一的参数是lpServiceClassInfo，它指向一个WSASERVICECLASSINFO结构，这个结构定义了这个服务类的属性。它的格式如下：

```
typedef struct _WSAServiceClassInfo {
    LPGUID              lpServiceClassId;
    LPTSTR              lpszServiceClassName;
    DWORD               dwCount;
    LPWSANSCLASSINFO    lpClassInfos;
} WSASERVICECLASSINFO, *PWSASERVICECLASSINFO, LPWSASERVICECLASSINFO;
```

第一个字段是GUID，它对这个特殊的服务类进行唯一性识别。要在这里使用 GUID，有两种方法可生成它。一是利用公用程序 Uuidgen.exe，并为这个服务器类建立一个 GUID。采用这种方法的问题就是：如果需要再次引用这个 GUID，就必须把它的值硬编码到头文件中的

某个地方。鉴于这一不便，第二种方法应运而生。在头文件 Svcguid.h内，有几个宏可生成基于一个简单属性的GUID。比方说，如果你安装SAP服务类（将声明你的IPX应用程序），就可用SVCID_NETWARE宏。唯一的参数便是你为自己的应用程序类分配的 SAP ID编号。SAP ID编号是在NetWare中预先定义好的，比如说，0x4表示文件服务器，而0x7则表示打印服务器。若采用后一种方法，只需一个易于记忆的 SAP ID，利用它生成指定服务类的GUID。另外，有几个宏把端口号当作一个参数来接收，并返回相应服务的 GUID。现在来看看头文件 Svcguide.h，其中包含一些针对逆向操作的宏，这些宏都是有用的 —— 从GUID中取出服务端口号。GUID是利用宏通过诸如端口号或SAP ID之类的属性生成的，表10-2列出了其中最常用的几个宏。头文件中还包括了一些众所周知的端口号（为 FTP和Telnet之类的服务预留的）的常量。

表10-2 常用的服务ID宏

宏	说 明
SVCID_TCP（Port）	通过TCP端口号生成一个GUID
SVCID_DNS（RecordType）	通过一个DNS记录类型生成一个GUID
SVCID_UDP（Port）	通过UDP端口号生成一个GUID
SVCID_NETWARE（SapId）	通过SAP ID编口号生成一个GUID

WSASERVICECLASSINFO结构的第二个字段是lpszServiceClassName，它仅仅是这个特定服务类的一个字串名。最后两个字段是描述性的： dwCount字段引用一个数值，表示投递给lpClassInfos字段的WSANSCLASSINFO结构有多少，这些结构对事实上服务所用的名字空间和协议特征进行定义。事实上的服务是指在这个服务类下注册的服务。它的格式如下：

```
typedef struct _WSANSClassInfo {
    LPSTR   lpszName;
    DWORD   dwNameSpace;
    DWORD   dwValueType;
    DWORD   dwValueSize;
    LPVOID  lpValue;
}WSANSCLASSINFO, *PWSANSCLASSINFO, *LPWSANSCLASSINFO;
```

lpszName字段定义服务类处理属性。表 10-3列出了可用的各种属性。其中每个属性都有一个REG_DWORD类的值。

表10-3 服务类型

字 串 值	定义的常量	名字空间	说 明
“SapId”	SERVICE_TYPE_VALUE_SAPID	NS_SAP	SAP ID
“ConnectionOriented”	SERVICE_TYPE_VALUE_CONN	任何一种	指明服务是面向连接的，还是无连接的
“TcpPort”	SERVICE_TYPE_VALUE_TCPPORT	NS_DNS NS_NTDS	TCP端口
“UdpPort”	SERVICE_TYPE_VALUE_UDPPORT	NS_DNS NS_NTDS	UDP端口

dwNameSpace是这个属性所用的名字空间。表 10-3列出了各种服务类型通常使用的名字空间。后三个字段dwValueType、dwValueSize和lpValue，都对真正与服务类型关联的那个值进行了描述。dwValueType字段标识与这个条目关联的数据的类型，所以称之为注册类型值。比如，这个值如果是DWORD，其类型就应该是REG_DWORD。下一个字段dwValueSize，仅

仅是被当作lpValue投递的数据之长度，lpValue是一个数据指针。

至于如何安装一个名为“Widget Server Class”的服务类，下面的代码示例对此进行了解释：

```
WSASERVICECLASSINFO    sci;
WSANSCLASSINFO        aNameSpaceClassInfo[4];
DWORD             dwSapId = 200,
                 dwUdpPort = 5150,
                 dwZero = 0;
int              ret;

memset(&sci, 0, sizeof(sci));

SET_NETWARE_SVCID(&sci.lpServiceClassId, dwSapId);
sci.lpszServiceClassName = (LPSTR)"Widget Server Class";
sci.dwCount = 4;
sci.lpClassInfos = aNameSpaceClassInfo;

memset(aNameSpaceClassInfo, 0, sizeof(WSANSCLASSINFO) * 4);
// NTDS name space setup
aNameSpaceClassInfo[0].lpszName = SERVICE_TYPE_VALUE_CONN;
aNameSpaceClassInfo[0].dwNameSpace = NS_NTDS;
aNameSpaceClassInfo[0].dwValueType = REG_DWORD;
aNameSpaceClassInfo[0].dwValueSize = sizeof(DWORD);
aNameSpaceClassInfo[0].lpValue = &dwZero;

aNameSpaceClassInfo[1].lpszName = SERVICE_TYPE_VALUE_UDPPORT;
aNameSpaceClassInfo[1].dwNameSpace = NS_NTDS;
aNameSpaceClassInfo[1].dwValueType = REG_DWORD;
aNameSpaceClassInfo[1].dwValueSize = sizeof(DWORD);
aNameSpaceClassInfo[1].lpValue = &dwUdpPort;

// SAP name space setup
aNameSpaceClassInfo[2].lpszName = SERVICE_TYPE_VALUE_CONN;
aNameSpaceClassInfo[2].dwNameSpace = NS_SAP;
aNameSpaceClassInfo[2].dwValueType = REG_DWORD;
aNameSpaceClassInfo[2].dwValueSize = sizeof(DWORD);
aNameSpaceClassInfo[2].lpValue = &dwZero;

aNameSpaceClassInfo[3].lpszName = SERVICE_TYPE_VALUE_SAPID;
aNameSpaceClassInfo[3].dwNameSpace = NS_SAP;
aNameSpaceClassInfo[3].dwValueType = REG_DWORD;
aNameSpaceClassInfo[3].dwValueSize = sizeof(DWORD);
aNameSpaceClassInfo[3].lpValue = &dwSapId;

ret = WSAInstallServiceClass(&sci);
if (ret == SOCKET_ERROR)
{
    printf("WSAInstallServiceClass() failed %d\n", WSAGetLastError());
}
```

首先要注意的是，这段代码采用了一个 GUID，上面的服务类将在这个 GUID下注册。你设计的服务都属于Widget Server Class这个类，而这个服务类描述的则是常见的属性，这些属性又属于一个服务实例。这个示例中，我们选用 NetWareSAP ID of 200来注册服务类。这样做

只是为了方便。其实，应该用一个任意的 GUID或基于UDP端口号的GUID。另外，当客户机正在端口5150上监听时，该服务可以使用 UDP协议。

下一个注意点是WSASERVICECLASSINFO结构的dwCount字段被设为4。这个示例中，将用SAP名字空间（NS_SAP）和Windows NT 域名空间（NS_NTDS）来注册服务类。其余需要注意的是：即使是在只用两个名字空间注册这个服务类，但这里却用了四个 WSANSCLASSINFO结构。因为我们为每个名字空间都定义了两个属性，而每个属性都需要一个独立的WSANSCLASSINFO结构。为每个名字空间定义了服务是否面向连接。这个示例中，名字空间是无连接的，因为我们把 SERVICE_TYPE_VALUE_CONN的值设成了一个布尔值 0。我们还针对Windows NT域名空间，通过使用服务类型SERVICE_TYPE_VALUE_UDPPORT，设置了UDP端口号，这个服务一般在这个端口下运行。针对SAP名字空间，我们用服务类型SERVICE_TYPE_VALUE_SAPID设置了自己的服务SAP ID。

针对每一个WSANSCLASSINFO条目，在设置服务类型和值长度时，还必须设置名字空间识别符，服务类型将在应用于这一名字空间。表 10-3中包括服务类型所需的类型，这一示例中结果都是DWORD。最后一步是简单调用 WSAInstallServiceClass，并把它当作一个参数投给WSASERVICECLASSINFO结构。如果WSAInstallServiceClass调用成功，就返回0；反之，则返回 SOCKET_ERROR。如果 WSASERVICECLASSINFO无效或排列有误，WSAGetLastError就会返回WSAEINCAL。如果这个服务类已经存在， WSAGetLastError则返回WSAEALREADY。这种情况下，就可以调用WSARemoveServiceClass，删掉一个服务器类。它的声明如下：

```
INT WSARemoveServiceClass( LPGUID lpServiceClassId );
```

这个函数的唯一参数是指向GUID的指针。这个GUID就是定义具体服务类的GUID。

10.3.2 服务的注册

服务类一旦安装（说明你的服务有哪些常见属性），就可以注册自己的服务实例，这样一来，远程机器上的其他客户机就可使用这一服务了。注册一个服务实例的 Winsock函数是WSASetService。

```
INT WSASetService (
    LPWSAQUERYSET lpqsRegInfo,
    WSAESETSERVICEOP essOperation,
    DWORD dwControlFlags
);
```

第一个参数lpqsRegInfo，是指向WSAQUERYSET结构的指针，该指针定义特定的服务。我们简要说明这个结构。 essOperation参数指定即将发生的行为，比如注册或取消注册。表 10-4对这三个有效标志进行了说明。

第三个参数dwControlFlags，不是0就是SERVICE_MULTIPLE这个标志。如果要在具体的服务实例下注册若干个地址，用这个标志即可。比如，你有一个服务，想在五台机器上运行它。投入WSAService的WSAQUERY结构将引用五个CSADDR_INFO结构，一一对该服务的实例进行描述。这时便需要设置 SERVICE_MULTIPLE标志。稍后，可取消注册一个单一的服务实例，这是通过利用 RNRSERVICE_DELETE标志来完成的。表10-5给出了操作和控制标志的可能组合，并根据服务的存在与否，对命令结果进行了描述。

表10-4 设置服务标志

操作标志	含 义
RNRSERVICE_REGISTER	注册一个服务名。对动态名字提供者而言，这个标志的意思是开始动态地声明这个服务。对固定的名字提供者来说，无意义
RNRSERVICE_DEREGISTER	从注册表中删除整个服务。对动态名字提供者而言，意味着中止声明服务。对固定的名字提供者来说，意味着从数据库中删除这个服务。对静态名字提供者来说，无意义
RNRSERVICE_DELETE	只将具体的服务实例从注册表中删除。一个注册服务可能包含若干个实例（这是在注册之后，紧接着利用 SERVICE_MULTIPLE标志来完成的）。再次提醒大家注意，这个标志只能应用于动态和固定名字提供者

表10-5 WSASetService标志的组合

RNRSERVICE_REGISTER

标 志	含 义	
	若服务已存在	若服务不存在
none	覆盖现成的服务实例	在具体的地址上增加一条新的服务条目
SERVICE_MULTIPLE	通过增加新地址的方式，更新服务实例	在具体的地址上增加一条新的服务条目

RNRSERVICE_DEREGISTER

标 志	含 义	
	若服务已存在	若服务不存在
none	删除所有的服务实例，但不删除服务（一般）说来，是保留WSAQUERY, 但CSADDR_INFO结构数是0	这是一个错误，将返回WSASERVICE_NOT_FOUND
SERVICE_MULTIPLE	通过删除具体地址的方式，对这个服务进行更新。即便无地址，这个服务仍然会存在	这是一个错误，并返回WSASERVICE_NOT_FOUND

RNRSERVICE_DELETE

标 志	含 义	
	若服务已存在	若服务不存在
none	把这个服务彻底从名字空间删除	这是一个错误，并返回 WSASERVICE_NOT_FOUND
SERVICE_MULTIPLE	通过删除具体地址的方式，对这个服务进行更新。如果不保留地址，服务就会彻底从名字空间删除	这是一个错误，并返回WSASERVICE_NOT_FOUND

至此，大家已知道了WSASetService的用法。接下来看看WSAQUERYSET结构。这个结构需要填充并投入函数。它的格式如下：

```
typedef struct _WSAQuerySetW {
    DWORD          dwSize;
    LPTSTR         lpszServiceInstanceName;
    LPGUID         lpServiceClassId;
    LPWSAVERSION   lpVersion;
    LPTSTR         lpszComment;
    DWORD          dwNameSpace;
    LPGUID         lpNSProviderId;
    LPTSTR         lpszContext;
    DWORD          dwNumberOfProtocols;
```

```
    LPAFPROTOCOLS    lpafpProtocols;
    LPTSTR           lpszQueryString;
    DWORD            dwNumberOfCsAddrs;
    LPCSADDR_INFO    lpcsaBuffer;
    DWORD            dwOutputFlags;
    LPBLOB           lpBlob;
} WSAQUERYSETW, *PWSAQUERYSETW, *LPWSAQUERYSETW;
```

dwSize字段应该设为WSAQUERYSET结构的长度。lpszServiceInstanceName字段中包含一个名字服务实例的字串识别符。lpServiceClassId字段是服务实例所属的那个服务类的GUID。lpVersion字段是可选的。在客户机请求一个服务时，可利用它获得有用的版本信息。lpszComment字段也是可选的。可在这里指定一个任何类型的注释字串。 dwNameSpace字段指定准备用来注册服务的名字空间。如果你只用一个名字空间，就只能用那个值；反之，就用NS_ALL。另外，可以引用一个定制的名字空间提供者（关于怎样编写自己的名字空间，将在第14章论述）。在定制的命名空间提供者这种情况下， dwNameSpace字段设为0，而lpNSProviderId指定代表定制名字空间提供者的GUID。lpszContext字段指定分层式名字空间（NDS ）中的查询起点。

dwNumberOfProtocols和lpafpProtocols字段都是可选的参数，用于限制搜索，只返回所提供协议。dwNumberOfProtocols字段对AFPROTOCOLS结构的数目进行引用，这些结构包含在lpafpProtocols数组中。它的格式如下：

```
typedef struct _AFPROTOCOLS {
    INT iAddressFamily;
    INT iProtocol;
} AFPROTOCOLS, *PAFPROTOCOLS, *LPAFPROTOCOLS;
```

第一个参数iAddressFamily，是AF_INET和AF_IPX之类的地址家族常量。第二个参数iProtocol是源于指定地址家族的协议，比如IPPROTO_TCP和NSPROTO_IPX。

WSAQUERYSET结构中的下一个字段lpszQueryString是可选的，只供支持丰富结构化查询语言（SQL）的名字空间所用，比如说Whois++。这个参数用来指定字串。

注册一个服务时，接下来的这两个参数是最重要的。 dwNumberOfCsAddrs字段只提供了投到lpcsaBuffer中的CSADDR_INFO结构的数目。CSADDR_INFO结构定义地址家族和服务事实上定位的地址。如果出现多个结构，就可以用多个服务实例。该结构的定义如下：

```
typedef struct _CSADDR_INFO {
    SOCKET_ADDRESS LocalAddr;
    SOCKET_ADDRESS RemoteAddr;
    INT            iSocketType;
    INT            iProtocol;
} CSADDR_INFO;

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

这一结构中还还包括了SOCKET_ADDRESS的定义。在注册一个服务时，可指定本地和远程地址。本地地址字段（LocalAddr）用于指定这个服务实例应该绑定的地址，而远程地址字段（RemoteAddr）则用于指定客户机在connect或sendto调用中，应该使用的那个地址。其他两个字段（ iSocketType和iProtocol）则是为具体的地址指定套接字类型（比方说

SOCK_STREAM和SOCK_DGRAM）和协议家族（比方说AF_INET和AF_IPX）。

WSAQUERYSET结构的最后两个字段是dwOutputFlags和lpBlob。一般说来，服务注册不需要这两个字段；不过在查询服务实例时，它们是非常有用的（将在下一节讨论）。只有名字空间提供者才可以返回一个 BLOB结构。也就是说，在注册一个服务时，你不能增加自己的BLOB结构，令其在客户机查询中返回。

表10-6列出了WSAQUERYSET结构的字段，并根据执行查询还是注册，判断哪些字段是要求的，哪些又是可选的。

表10-6 WSAQUERYSET字段

字 段	查 询	注 册
dwSize	要求	要求
lpszServiceInstanceName	要求字串或“*”	要求
lpServiceClassId	要求	要求
lpVersion	可选	可选
lpszComment	忽略	可选
dwNameSpace	二选一	二选一
lpNSProviderId	必须指定的字段	必须指定字段
lpszContext	可选	可选
dwNumberOfProtocols	0或0以上	0或0以上
lpafpProtocols	可选	可选
lpszQueryString	可选	忽略
dwNumberOfCsAddrs	忽略	要求
lpcsaBuffer	忽略	要求
dwOutputFlags	忽略	可选
lpBlob	忽略，可通过查询返回	忽略

10.3.3 服务注册示例

这一小节中，将向大家展示如何在 SAP和NTDS这两个名字空间下注册自己的服务。Windows NT域名空间相当有用，这就是我们把它包含到示例中的原因。尽管如此，必须了解它的下面这些特性。首先，Windows NT域名空间需要Windows 2000，因为它是以“活动目录”（Active Directory）为基础的。另外，对你打算在上面注册和(／或)查找服务的Windows 2000工作站而言，还意味着必须有一个账号，可以在这个域内访问“活动目录”。另一个需要注意的特性是Windows NT域名空间能够注册源于任何一个协议家族的套接字地址。这意味着你的IP和IPX服务均可以注册在同一个名字空间内。程序清单 10-1解释了注册一个服务实例的所需要的基本步骤。为了简单起见，代码中没有执行错误检查。

程序清单10-1 WSASetService示例

```
SOCKET              socks[2];
WSAQUERYSET         qs;
CSADDR_INFO         lpCSAddr[2];
SOCKADDR_IN         sa_in;
SOCKADDR_IPX        sa_ipx;
IPX_ADDRESS_DATA    ipx_data;
GUID                guid = SVCID_NETWARE(200);
int                 ret, cb;
```

```
memset(&qs, 0, sizeof(WSAQUERYSET));
qs.dwSize = sizeof(WSAQUERYSET);
qs.lpszServiceInstanceName = (LPSTR)"Widget Server";
qs.lpServiceClassId = &guid;
qs.dwNameSpace = NS_ALL;
qs.lpNSProviderId = NULL;
qs.lpcsaBuffer = lpCSAddr;
qs.lpBlob = NULL;
//
// Set the IP address of our service
//
memset(&sa_in, 0, sizeof(sa_in));
sa_in.sin_family = AF_INET;
sa_in.sin_addr.s_addr =  htonl(INADDR_ANY);
sa_in.sin_port = 5150;

socks[0] = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
ret = bind(socks[0], (SOCKADDR *)&sa_in, sizeof(sa_in));

cb = sizeof(sa_in);
getsockname(socks[0], (SOCKADDR *)&sa_in, &cb);

lpCSAddr[0].iSocketType = SOCK_DGRAM;
lpCSAddr[0].iProtocol = IPPROTO_UDP;
lpCSAddr[0].LocalAddr.lpSockaddr = (SOCKADDR *)&sa_in;
lpCSAddr[0].LocalAddr.iSockaddrLength = sizeof(sa_in);
lpCSAddr[0].RemoteAddr.lpSockaddr = (SOCKADDR *)&sa_in;
lpCSAddr[0].RemoteAddr.iSockaddrLength = sizeof(sa_in);
//
// Set up the IPX address for our service
//
memset(sa_ipx.sa_netnum, 0, sizeof(sa_ipx.sa_netnum));
memset(sa_ipx.sa_nodenum, 0, sizeof(sa_ipx.sa_nodenum));
sa_ipx.sa_family = AF_IPX;
sa_ipx.sa_socket = 0;

socks[1] = socket(AF_IPX, SOCK_DGRAM, NSPROTO_IPX);

ret = bind(socks[1], (SOCKADDR *)&sa_ipx, sizeof(sa_ipx));

cb = sizeof(IPX_ADDRESS_DATA);
memset (&ipx_data, 0, cb);
ipx_data.adapternum = 0;

ret = getsockopt(socks[1], NSPROTO_IPX, IPX_ADDRESS,
    (char *)&ipx_data, &cb);

cb = sizeof(SOCKADDR_IPX);
getsockname(socks[1], (SOCKADDR *)sa_ipx, &cb);

memcpy(sa_ipx.sa_netnum, ipx_data.netnum, sizeof(sa_ipx.sa_netnum));
memcpy(sa_ipx.sa_nodenum, ipx_data.nodenum, sizeof(sa_ipx.sa_nodenum));
lpCSAddr[1].iSocketType = SOCK_DGRAM;
lpCSAddr[1].iProtocol = NSPROTO_IPX;
lpCSAddr[1].LocalAddr.lpSockaddr = (struct sockaddr *)&sa_ipx;
lpCSAddr[1].LocalAddr.iSockaddrLength = sizeof(sa_ipx);
lpCSAddr[1].RemoteAddr.lpSockaddr = (struct sockaddr *)&sa_ipx;
```

```
lpCSAddr[1].RemoteAddr.iSockaddrLength = sizeof(sa_ipx);

qs.dwNumberOfCsAddrs = 2;

ret = WSASetService(&qs, RNRSERVICE_REGISTER, 0L);
```

程序清单10-1中的示例代码解释了如何设置一个服务实例，这样一来，该服务的一个客户机便可找到自己需要与之通信的服务地址。第一个步骤是初始化 WSAQUERY结构。另外，我们还需要为自己的服务实例名字。这里，我们简单地把它叫作 WidgetServer。另一个关键性步骤是利用注册服务类时所用的一个 GUID。无论何时注册一个服务实例，这个服务都必须属于一个服务类。这里，我们用的是“ WidgetService Class ”（前一小节定义的），它的GUID是SVCID_NETWARE(200)。下一步便是设置我们感兴趣的名字空间。由于我们的服务运行于IPX和UDP，因而指定的是 NS_ALL。因为我们指定的是先前存在的名字空间，所以必须把lpNSProviderId设为NULL。

下一步是设置CSADDR_INFO数组内的SOCKADDR结构。WSASetService把这个结构当作WSAQUERYSET结构中的lpcsaBuffer字段投递。大家将注意到，我们的示例中的确建立了套接字，并在设置SOCKADDR结构之前，把它们绑定到一个本地地址。这是因为我们必需找到客户机打算连接的那个确切的本地地址。比方说，在为服务器建立我们的 UDP时，我们绑定了INADDR_ANY，直到调用 getsocketname，才会把事实上的 IP地址给我们。利用getsocketname返回的信息，我们就可建立SOCKADDR_IN结构了。在CSADDR_INFO结构内，我们设置了套接字类型和协议。另两个字段是本地和远程地址信息。本地地址是服务器应该绑定的地址，而远程地址则是客户机应该用来连接服务的地址。

在为这个基于UDP的服务器设置SOCKADDR_INFO结构之后，我们设置了基于 IPX的服务。在第6章中，大家已知道服务器应该和网络编号绑定在一起，这个网络编号是通过把网络和节点编号设为0而得的。再次提醒大家注意，它并没有给出客户机所需的那个地址，因此，需要调用套接字选项IPX_ADDRESS来获得事实上的地址。在填充 IPX的CSADDR_INFO结构时，分别利用套接字类型和协议的 SOCK_DGRAM和NSPROTO_IPX即可。最后一步是把WSAQUERYSET结构中的dwNumberOfAddrs字段设为2，因为有两个地址（UDP和IPX）客户机用它们来建立一个连接。最后，调用 WSASetSetService函数，它带有WSAQUERYSET结构和RNRSERVICE_REGISTER标志，但没有控制标志。不要指定 SERVICE_MULTIPLE控制标志，这样，在选择注册我们的服务时，这个服务的所有实例（ IPX和UDP地址）都会被取消注册。

另外，需要注意的是：上面的示例中没有考虑多址机。如果你在一台多址机上建立一个绑定INADDR_ANY的基于UDP的服务器，客户机就可以在任何一个接口上与这个服务器建立连接。在IP的情况下，getsockname是不充分的；你必须获得所有的本地 IP接口。获得这类信息的方法要根据各人使用的平台而定。对所有平台而言，常见的方法是调用 gethostbyname，返回一张我们名字的IP地址列表。在Winsock 2下，还可以调用ioctl命令SIO_GET_INTERFACE_LIST。针对Windows 2000平台，可使用ioctl命令SIO_ADDRESS_LIST_QUERY。最后，还可以用附录 B中讨论的IP助手函数。简单的 TCP/IP名字解析和gethostbyname的介绍参见第6章，而ioctl命令则参见第9章。除此以外，附带光盘上的 Rnrcs.c是一个完整翔实的、以多址机为重点的示例。

10.4 服务的查询

现在，大家已经知道如何在名字空间内注册服务了，下面来看看客户机是如何向名字空间查询具体服务，进而获得通信服务的有关信息的。尽管名字解析采用的查询函数有三个之多：WSALookupServiceBegin、WSALookupServiceNext和WSALookupServiceEnd，但和服务注册比起来，仍然要简单些。执行查询的第一步是调用 WSALookupServiceBegin，这个函数通过设置查询限制来开始查询。它的原型如下：

```
INT WSALookupServiceBegin (
    LPWSAQUERYSET lpqsRestrictions,
    DWORD dwControlFlags,
    LPHANDLE lphLookup
);
```

第一个参数是一个WSAQUERY结构，它把限制加到查询上，比如说限定查询哪些名字空间。第二个参数dwControlFlags，决定查询的深度。表10-7中包含了各种可能用到的标志及其含义。这些标志都会影响查询的行为和查询返回的数据。最后一个参数是 HANDLE类型的，在函数返回之后利用。若成功，返回的值就是 0；否则，就返回SOCKET_ERROR。如果一个或一个以上的参数无效， WSAGetLastError就会返回WSAEINVAI。如果在名字空间内找到该服务名，但没有可与所给限制匹配的数据，错误就是 WSANO_DATA。若指定服务不存在，则返回WSASERVICE_NOT_FOUND错误。

表10-7 控制标志

标 志	含 义
LUP_DEEP	在分层式名字空间中，请求深度与第一级相对应
LUP_CONTAINERS	只获得容器对象。该标志只适合分层式名字空间
LUP_NOCONTAINERS	不返回任何容器。该标志只适合分层式名字空间
LUP_FLUSHCACHE	忽略隐藏信息，直接查询名字空间。注意并非所有的名字提供者都隐藏查询
LUP_FLUSHPREVIOUS	令名字提供者丢弃前一次返回的信息集。这个标志一般在 WSALookupServiceNext返回WSA_NOT_ENOUGH_MEMORY之后才用。如果所提供的缓冲区不够，信息就会被丢弃，并检索下一个信息集
LUP_NEAREST	按距离顺序检索结果。注意，这由计算距离公制的名字提供者来决定，因为注册服务时没有为该信息作准备。不要求名字提供者支持这一概念
LUP_RES_SERVICE	指定本地地址在CSADDR_INFO结构中返回
LUP_RETURN_ADDR	获得lpcsaBuffer形式的地址
LUP_RETURN_ALIASES	只获得别名信息。每个别名都会在成功调用 WSALookupService中返回
LUP_RETURN_ALL	获得所有有用的信息
LUP_RETURN_BLOB	获得lpBlob形式的私有数据
LUP_RETURN_COMMENT	获得lpszComment形式的注释
LUP_RETURN_NAME	获得lpszServiceInstanceName形式的名字
LUP_RETURN_TYPE	获得lpServiceClassId形式的类型
LUP_RETURN_VERSION	获得lpVersion形式的版本号

在调用WSALookupServiceBegin时，返回的查询句柄是你投给WSALookupServiceNext的，它会返回你需要的数据。该函数的定义如下：

```
INT WSALookupServiceNext (
    HANDLE hLookup,
    DWORD dwControlFlags,
    LPDWORD lpdwBufferLength,
```

```
    LPWSAQUERYSET lpqsResults
);
```

句柄hLookup通过WSALookupServiceBegin返回。对dwControlFlags参数而言，除了只支持LUP_FLUSHPREVIOUS外，其含义和 WSALookupServiceBegin是相同的。参数lpdwBufferLength被当作lpqsResults投递，是一个缓冲区长度。由于 WSAQUERYSET结构中应该包含二进制的大型对象（ BLOB）数据。它通常要求你投递一个大于结构本身的缓冲区。对即将返回的数据而言，如果提供的缓冲区长度不够，函数调用就会失败，并出现WSA_NOT_ENOUGH_MEMORY错误。

一旦利用WSALookupServiceBegin开始查询，就要在生成 WSA_E_NO_MORE(10110)之前，调用WSALookupServiceNext。特别需要注意的是：早期的Winsock实施中，“ 无过多数据 ”的错误代码是WSAENOMORE(10102)，因此，对一个健壮的代码而言，应该还要对这两个错误代码都进行检查。一旦所有数据都返回，或已经完成查询，就调用查询过程中所用的、带有HANDLE变量的WSALookupServiceEnd。该函数的定义如下：

```
INT WSALookupServiceEnd ( HANDLE hLookup );
```

10.4.1 怎样对服务进行查询

如何对前一小节中注册的服务进行查询呢？第一件事是设置定义查询所用的 WSAQUERY结构。我们来看看下面的代码：

```
WSAQUERYSET    qs;
GUID        guid = SVCID_NETWARE(200);
AFPROTOCOLS      afp[2] = {{AF_IPX,  NSPROTO_IPX}, {AF_INET, IPPROTO_UDP}};
HANDLE        hLookup;
int         ret;

memset(&qs, 0, sizeof(qs));
qs.dwSize = sizeof (WSAQUERYSET);
qs.lpszServiceInstanceName = "Widget Server";
qs.lpServiceClassId = &guid;
qs.dwNameSpace = NS_ALL;
qs.dwNumberOfProtocols = 2;
qs.lpafpProtocols = afp;
ret = WSALookupServiceBegin(&qs, LUP_RETURN_ADDR | LUP_RETURN_NAME,
         &hLookup);
if (ret == SOCKET_ERROR)
    // Error
```

记住，所有的服务查询都在服务类 GUID的基础上进行，你查询的服务就是在这个服务类基础上注册的。变量 guid设为服务器的服务类ID。先把qs初始化成0，在把dwSize字段设为结构的长度。下一步是给出准备查询的服务名。服务名可以是服务的真名，也可以指定一个通配（*），通配将返回具体的服务类 GUID的所有服务。接下来，要求查询利用 NS_ALL常量搜索所有的名字空间。最后，设置协议（ IPX和UDP/IP），以便我们的客户机能与之建立连接。这是利用一个由两个AFPROTOCOLS结构组成的数组来完成的。

现在，准备开始查询，首先必须调用 WSALookupServiceBegin。它的第一个参数是我们的WSAQUERYSET结构，而后面的几个参数则是标志，这些标志定义找到相符的服务时应该返回呢数据。这里，你应该对 LUP_RETURN_ADDR和LUP_RETURN_NAME这两个标志执

行按位和运算，藉此说明自己需要定址和服务名；只有在用通配符（*）来指代服务名时，才需要LUP_RETURN_NAME标志；否则就是已经知道服务名了。最后一个参数是一个识别这个特殊查询的HANDLE变量。成功返回之后，它就会被初始化。

一旦成功打开查询，就可在WSA_E_NO_MORE返回之前，调用WSALookupServiceNext。每个成功调用都会返回符合查询标准的那个服务的相关信息。这一过程的代码如下：

```
char           buff[sizeof(WSAQUERYSET) + 2000];
DWORD          dwLength, dwErr;
WSAQUERYSET   *pqs = NULL;
SOCKADDR      *addr;
int            i;

pqs = (WSAQUERYSET *)buff;
dwLength = sizeof(WSAQUERYSET) + 2000;
while (1)
{
    ret = WSALookupServiceNext(hLookup, 0, &dwLength, pqs);
    if (ret == SOCKET_ERROR)
    {
        if ((dwErr = WSAGetLastError()) == WSAEFAULT)
        {
            printf("Buffer too small; required size is: %d\n", dwLength);
            break;
        }
        else if ((dwErr == WSA_E_NO_MORE) || (dwErr = WSAENOMORE))
            break;
        else
        {
            printf("Failed with error: %d\n", dwErr);
            break;
        }
    }
    for (i = 0; i < pqs->dwNumberOfCsAddrs; i++)
    {
        addr = (SOCKADDR *)pqs->lpcsaBuffer[i].RemoteAddr.lpSockaddr;
        if (addr->sa_family == AF_INET)
        {
            SOCKADDR_IN *ipaddr = (SOCKADDR_IN *)addr;
            printf("IP address:port = %s:%d\n", inet_ntoa(addr->sin_addr),
            addr->sin_port);
        }
        else if (addr->sa_family == AF_IPX)
        {
            SOCKADDR_IPX *ipxaddr = (SOCKADDR_IPX *)addr;
            printf("%02X%02X%02X%02X.%02X%02X%02X%02X%02X%02X:%04X",
                (unsigned char)ipxaddr->sa_netnum[0],
                (unsigned char)ipxaddr->sa_netnum[1],
                (unsigned char)ipxaddr->sa_netnum[2],
                (unsigned char)ipxaddr->sa_netnum[3],
                (unsigned char)ipxaddr->sa_nodenum[0],
                (unsigned char)ipxaddr->sa_nodenum[1],
                (unsigned char)ipxaddr->sa_nodenum[2],
                (unsigned char)ipxaddr->sa_nodenum[3],
                (unsigned char)ipxaddr->sa_nodenum[4],
                (unsigned char)ipxaddr->sa_nodenum[5],
```

```
                ntohs(ipxaddr->sa_socket));
            }
        }
    }
    WSALookupServiceEnd(hLookup);
```

尽管这段示例代码有点简单，但非常明晰。调用 WSALookupServiceNext只需要一个有效的查询句柄、返回缓冲区的长度和返回缓冲区本身。不需要指定任何控制标志，因为唯一可用于该函数标志只有一个：LUP_FLUSHPREVIOUS。如果我们提供的缓冲区太小，若设置这个标志，该函数返回的结果就会被丢弃。然而，在我们的示例中，我们没有采用LUP_FLUSHPREVIOUS，如果我们提供的缓冲区太小，就会生成 WSAEFAULT错误。发生这种情况时，就要把lpdwBufferLength设为所需要的长度。我们的例子中采用了一个固定长度的缓冲区，相当于WSAQUERYSET结构再加2000个字节的长度。由于你只要求了所有的服务名和地址，所以这个长度绰绰有余。当然，如果要面向广大用户，你的应用程序中还应该能对WSAEFAULT错误进行处理。

一旦成功调用 WSALookupServiceNext，缓冲区内就会填入一个 WSAQUERYSET结构，这个结构中包含了返回的结果。我们的查询中，需要服务名和地址； WSAQUERYSET结构中，最重要的字段是 lpszServiceInstanceName和lpcsaBuffer。前者包含服务名，后者则是一个CSADDR_INFO结构数组，其中包含服务的定址信息。 dwNumberOfCsAddrs参数会准确地告诉我们已返回多少地址。上面的示例代码中，我们只是打印了地址。只检查并打印了 IPX和IP地址，因为在你打开查询时，只要求了这两个地址家族。

查询中，如果用通配符（*）来代表服务名，那么每次调用 WSALookupServiceNext，都会返回一个特定的服务实例，它运行于网络上某个地方——当然，前提是实际上已注册了多个服务实例，并且正处于运行状态。一旦服务的所有实例都已返回，就会产生WSA_E_NO_MORE错误，这样你就会中断循环。最后一件事是在查询句柄上调用WSALookupServiceEnd。这样便可释放为查询分配的所有资源。

10.4.2 查询DNS

前面，我们曾提过 DNS名字空间是静态的，这意味着你不能动态地注册自己的服务；但仍可用Winsock名字解析函数来执行DNS查询。实际上，执行DNS查询比执行普通的已注册服务查询要复杂得多，因为DNS名字空间提供者是以BLOB的形式返回查询信息。为什么会这样呢？还记得第6章中对gethostname的讨论吗？“名字检索返回的 HOSTNAME结构中不仅包含了IP地址，还包含了别名”。WSAQUERYSET结构偏偏就例外。

关于BLOB数据，需要注意的一点是：它的格式尚未很好地编入帮助文档，这样就使得直接查询DNS显得有些困难。我们首先来看看如何打开查询。本书附带光盘上的 Dnsquery.c文件包含了直接查询 DNS所用的完整的示例代码；我们逐段地对它们进行分析。下面的代码解释了如何初始化DNS查询：

```
WSAQUERYSET  qs;
AFPROTOCOLS  afp [2] = {{AF_INET, IPPROTO_UDP},{AF_INET, IPPROTO_TCP}};
GUID         hostnameguid = SVCID_INET_HOSTADDRBYNAME;
DWORD        dwLength = sizeof(WSAQUERYSET) + sizeof(HOSTENT) + 2048;
HANDLE       hQuery;

qs = (WSAQUERYSET *)buff;
```

```
memset(&qs, 0, sizeof(qs));
qs.dwSize = sizeof(WSAQUERYSET);
qs.lpszServiceInstanceName = argv[1];
qs.lpServiceClassId = &hostnameguid;
qs.dwNameSpace = NS_DNS;
qs.dwNumberOfProtocols = 2;
qs.lpafProtocols = afp;

ret = WSALookupServiceBegin(&qs, LUP_RETURN_NAME | LUP_RETURN_BLOB,
    &hQuery);
if (ret == SOCKET_ERROR)
    // Error
```

DNS查询的设置非常类似于我们的前一个例子。最显著的变化是我们这里采用的是 GUID SVCD_INET_HOSTADDRBYNAME。这是一个识别主机名查询的GUID。lpszServiceInstanceName是我们准备解析的主机名。由于正在通过 DNS解析主机名，所以我们只需为 dwNameSpace指定NS_DNS。最后，把lpafProtocols设成一个数组，该数组由两个 AFPROTOCOLS结构组成，它把TCP/IP和UDP/IP协议定义成我们的查询感兴趣的协议。

查询一旦建立，就可调用WSALookupServiceNext，返回数据：

```
char          buff[sizeof(WSAQUERYSET) + sizeof(HOSTENT) + 2048)];
DWORD         dwLength = sizeof(WSAQUERYSET) + sizeof(HOSTENT) + 2048;
WSAQUERYSET *pqs;
HOSTENT     *hostent;

pqs = (WSAQUERYSET *)buff;
pqs->dwSize = sizeof(WSAQUERYSET);
ret = WSALookupServiceNext(hQuery, 0, &dwLength, pqs);
if (ret == SOCKET_ERROR)
    // Error
WSALookupServiceEnd(hQuery);

hostent = pqs->lpBlob->pBlobData;
```

因为DNS名字空间提供者以BLOB的形式返回主机信息，所以需要你提供一个足够大的缓冲区。这就是为什么要用一个非常大的缓冲区的原因，其长度相当于一个 WSAQUERYSET加上一个HOSTENT，再加上2048个字节。再次提醒大家注意，如果长度还不够，函数调用就会失败，出现WSAEFAULT错误。在DNS查询中，所有的主机信息都返回在 HOSTENT结构内，即使主机名与多个IP地址关联在一起。这就是不必多次调用 WSALookupServiceNext的原因。

这个地方需要特别注意——对查询返回的BLOB结构进行解码。通过第 6章的学习，大家已知道HOSTENT结构的定义是这样的：

```
typedef struct hostent {
char FAR * h_name;
char FAR * FAR * h_aliases;
short h_addrtype;
short h_length;
char FAR * FAR * h_addr_list;
} HOSTENT;
```

HOSTENT结构以BLOB数据的形式返回时，结构内的指针对应的是实际是内存中的偏移位置，实际数据是自那个位置起开始存放的。这个偏移位置是自 BLOB数据的起始处开始算起的。这样一来，我们必须对指针作一番修正，使其指向绝对的内存位置，否则不能实际地访

问到数据。在图10-1中，我们向大家展示了HOSTENT结构以及返回的内存布局。DNS查询是主机名“ riven ”上执行的，该主机有一个对应的IP地址，但没有“ 别名 ”。结构中的每个字段都有一个偏移值。为纠正这个问题，使字段能引用到正确位置，我们需要将偏移值加到HOSTENT结构的头地址上。这一计算需要针对 h_name，h_aliases和h_addr_list字段进行。此外，h_aliases和h_addr_list字段指定的是一个指针数组。一旦取得了指向指针数组的正确指针，引用位置中的每个32位字段都会由偏移值构成。看看图 10-1展示的h_addr_list字段，便会发现初始偏移量是16字节，它指定的是 HOSTENT结构末尾之后的字节数。这是指向 4字节IP地址的一个指针数组。然而，数组中的每一个指针偏移距离是 28字节。要想令其指向正确的位置，需要先取得HOSTENT结构的地址，再在它的基础上加 28字节，指向一个实际的 4字节位置。在那个位置，含有数据 0x9D36B9BA，亦即IP地址157.54.185.186。随后，便可自那个位置开始，在第28个字节的偏移量之后，取得总长为 4个字节的数据（全部为 0）它标志着指针数组的结束。假如该主机名同时对应着几个 IP地址，那么必然存在着其他的偏移量。如法炮制，便能修正指针，得到正确的数据。针对 h_aliases指针以及它所引用的那个指针数组，也要采取同样的做法，将它们修正为正确的值。就本例来说，我们的主机并未设置别名。数组的第一个条目是0，表明不必再对那个字段采取任何多余的操作。最后一个字段是 h_name字段，非常容易纠正 —— 只需将偏移量加到HOSTENT结构地址上面就可以了，它指向的是一个空中止字串的起始处。

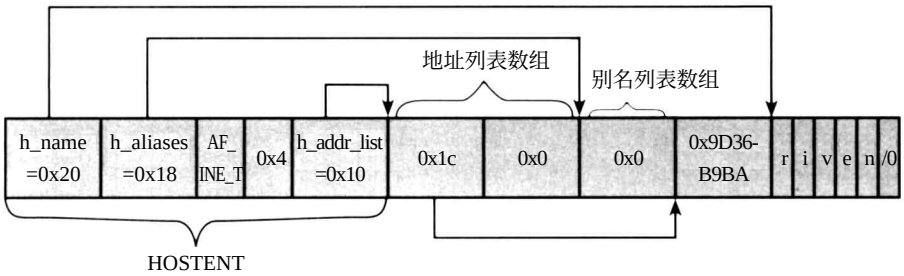

图10-1 HOSTENT BLOB数据

要把这些偏移修正为真正的地址，尽管涉及到大量的指针运算，但所需代码并不复杂。要修正h_name字段，进行简单的偏移调整即可，具体如下：

```
hostent->h_name = (PCHAR)((DWORD_PTR)hostent->h_name) + (PCHAR)hostent;
```

要修正指针数组（比如 h_aliases h_addr_list字段中的数组），需要的代码则要多一些，但也只需要在这个数组中完整地走一遍，依次修正每一个引用，直到碰到一个空条目为止。代码如下：

```
PCHAR  *addr;

if (hostent->h_aliases)
{
    addr = hostent->h_aliases = (PCHAR)((DWORD_PTR)hostent->h_aliases +
(PCHAR)hostent);
   while (addr)
   {
        addr = (PCHAR)((DWORD_PTR)addr + (PCHAR *)hostent);
        addr++;
```

```
        }
    }
```

这段代码只是逐步浏览了各个数组条目，把 HOSTENT结构的起始地址加到具体的偏移量上，这个偏移量即后来的条目的值。当然，一旦碰到了一个其值为 0的数组条目，你自然就会停下来。需要对h_addr_list字段如法炮制。一旦偏移得到了修正，就可以正常使用 HOSTENT结构了。

10.5 小结

RNR函数看起来过于复杂，但是，它们为编写客户机／服务器应用程序提供了非常大的灵活性。名字注册的真正局限在于名字空间。随着 TCP/IP的风行，DNS曾一度独领风骚，但DNS又欠灵活。直到有了Windows 2000和Windows NT域名空间，我们才有了一个稳定的、与协议无关的名字解析方法，它为编写强健的应用程序提供了充分的灵活性。另外，其他名字空间（比如SAP）也可用于以IPX/SPX为基础的应用程序，它们提供了许多类似于 NTDS的能力（与协议无关这一点除外）。