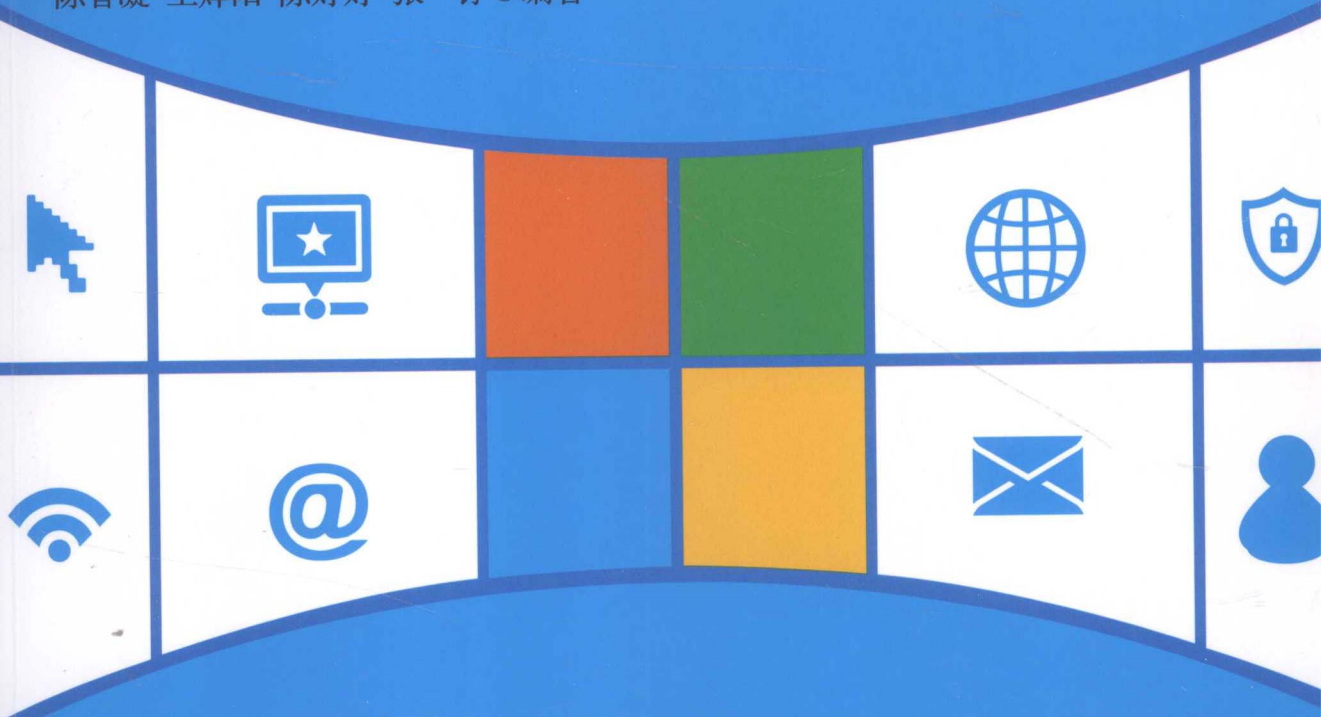
异步图书
www.epubit.com.cn

Windows 网络与通信程序设计

第3版

陈香凝 王烨阳 陈婷婷 张 铮◎编著



- 示例了 Winsock 各种 I/O 方法，详细说明了高性能可伸缩性服务器的开发过程，给出了详尽的实现代码。
- 将编程方法、网络协议和应用实例有机结合起来，详细介绍了 Internet 广播和 IP 多播、原始套节字、SPI、LAN 和 WAN 上的扫描和侦测技术、网络数据的窃取和保护、ARP 欺骗等。
- 详细演示了协议驱动的开发过程，介绍了 NDIS 编程接口。
- 一本让读者在编程实践中学习 P2P 程序设计的书，讨论了穿透防火墙、NAT 等直接建立 UDP 和 TCP 连接的各种方案。
- 包含了商业级 Windows 个人防火墙的完整实例代码，采用应用层（SPI）/ 核心层（IMD 驱动）双重过滤，完全管控 TCP/IP 网络封包。
- 涉及 60 多个完整实例，许多的例子稍做修改即可应用到实际项目中。

中国工信出版集团 人民邮电出版社 POSTS & TELECOM PRESS

第 1 章　计算机网络基础

本章详细讲述网络程序设计中要用到的计算机网络方面的基础知识，包括各种网络术语、网络硬件设备、网络拓扑结构、网络协议等。

1.1　网络的概念和网络的组成

网络是各种连在一起的可以相互通信的设备的集合。本书讲述的网络是最常见的，将数亿计算机连接到一起的 Internet。下面通过讲述组成 Internet 的基本硬件和软件来进一步明确计算机网络的概念。

Internet 是世界范围内的计算机网络，它不仅连接了 PC、存储和传输信息的服务器，还连接了 PDA、电视、移动 PC 等。所有的这些设备称为**主机**（host）或**终端系统**（end system）。

终端系统由**通信链接**（communication links）连在一起。常见的通信链接有双绞线、同轴电缆、光纤等，它们负责传递原始的比特流。

终端系统通常并不通过单一的通信链接相互连在一起，而是通过中介交换设备间接相连。这些中介交换设备称为**包交换器**（packet switch）。包交换器在通信链路上接收到达的信息块，并向其他的通信链路上推进这个信息块。这些信息块称为**包**（packet）。包交换器有多种形状和特色，当今 Internet 上最基本的两种包交换器是**路由器**（router）和**链路层交换器**（link-layer switch）。两种类型的交换器都推动包向它们的目的地址前进，后面还要详细地讨论它们。

从发送终端系统到接收终端系统，包所经过的通信链接和包交换器称为**路线**（route）或**路径**（path）。

每个终端系统通过 ISP（Internet Service Provider，Internet **服务提供商**）连接 Internet。ISP 拥有由许多通信链接和包交换器组成的网络，它提供的网络访问类型多种多样，有 56kbit/s 的拨号 Modem 访问、高速 LAN 访问、无线访问等。

终端系统、包交换器和 Internet 的其他部分，都运行**协议**（protocol）来控制数据的发送和接收，协议是计算机用来与其他计算机通信的语言。TCP（Transfer Control Protocol，**传输控制协议**）和 IP（Internet Protocol，**网际协议**）是两个最重要的协议。IP 指定了在路由器和终端系统中传输的封包的格式。Internet 中所有重要的协议共同称为 TCP/IP。本书还会详细介绍它们。

除了 Internet，还有许多专用网络，如许多公司和政府的网络。这些专用网络通常称为**企业内部互联网**（Intranet），它们使用的主机、路由器、链接和协议与 Internet 相同。

1.2　计算机网络参考模型

了解网络的相关概念之后，本节将讨论计算机网络中主机之间是如何进行通信的，以及各种通信协议之间的关系等。

1.2.1　协议层次

为了降低设计难度，大部分网络都以**层**（layer 或 level）的形式组织在一起，每一层都建立在它的下层之上，使用它的下层提供的服务，下层对它的上层隐藏了服务实现的细节。这种方法几乎应用于整个计算机科学领域，也可以称为信息隐藏、数据类型抽象、数据封装、面向对象编程等。

一个机器上的第 n 层和另一个机器的第 n 层交流，所使用的规则和协定合起来称为第 n 层协议。这里的**协议**，是指通信双方关于如何进行通信的一种约定。各层和各层协议的集合称为**网络体系**（network architecture）。特定系统所使用的一组协议称为**协议堆栈**（protocol stack）。下面介绍 Inernet 网络分层情况和它的协议堆栈。

1.2.2　TCP/IP 参考模型

为了帮助不同的厂商标准化和一体化它们的网络软件，1974 年，国际标准化组织（ISO，International Organization for Standardization）为在机器之间传送数据定义了一个软件模型，就是著名的 OSI 模型（Open Systems Interconnection，开放式系统互联模型）。这个模型共有 7 层，如图 1.1 所示。

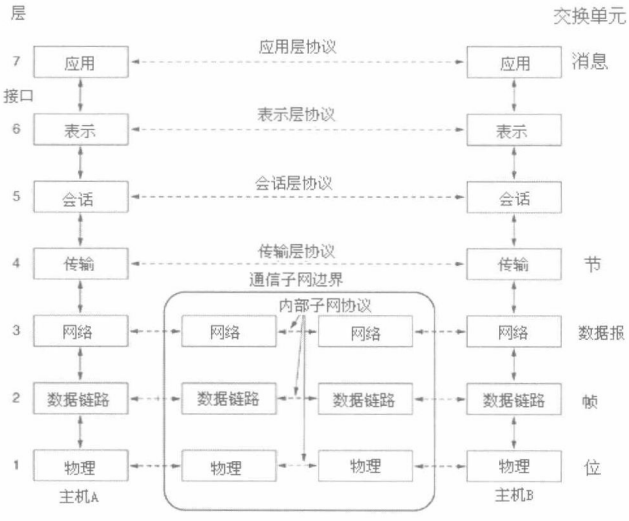


图 1.1　OSI 参考模型

OSI 参考模型仅是一个理想方案，几乎没有什么系统能够完全实现它，它存在的作用是给人们一个设计网络体系的框架。机器上的每一层都假设它正在直接与另一机器的同一

层“交谈”，它们“说”相同的语言，或者协议，各层的目的是向更高的层提供服务，抽象低层的实现细节。TCP/IP 实现了 OSI 参考模型中的 5 层，如图 1.2 所示，各层使用的协议连在一起便是互联网协议堆栈。

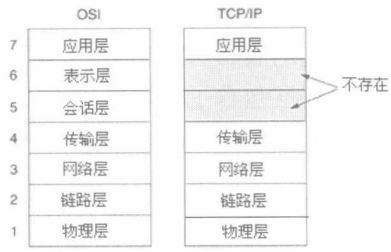


图 1.2 TCP/IP 与 OSI 参考模型

1.2.3 应用层（Application Layer）

应用层是网络应用程序和它们的应用层协议存在的地方。Internet 应用层包含许多协议，如 HTTP（它提供 Web 文档的请求和传输）、SMTP（它提供 e-mail 消息的传输）和 FTP（它提供两个终端系统间的文件传输）。一些特定的网络功能，如映射主机名到它们的网络地址的 DNS（Domain Name System，域名服务器）也在此层完成。

应用层程序设计在现实生活中应用最广泛，因为它是直接面向用户的。本书在后面要讨论的客户端和服务器端程序、P2P 通信程序等都属于此层。本书使用应用层消息来表示应用层的数据传输单元。

1.2.4 传输层（Transport Layer）

Internet 的传输层在应用程序的客户和服务器之间传递应用层消息，在这里定义了两个点对点的传输协议——TCP（Transmission Control Protocol，传输控制协议）和 UDP（User Datagram Protocol，用户数据报协议）。

TCP 是一个可靠的面向连接的协议，它允许源于一个机器的字节流被无错误地传输到 Internet 上的任何其他机器。TCP 将上层传递的字节流分成封包，再接着传递到它的下层——网络层。在接收方，TCP 重新集合接收到的封包，将其转化成为输出流。TCP 也处理流控制，以确保一个快的发送者不会发送太多的封包而淹没接收者。

UDP 是一个不可靠的无连接的协议，它是为那些不需要 TCP 的序列号管理和流控制，而想自己提供这些功能的应用程序设计的。

Windows 为传输层的编程接口提供了 Socket 函数，即通常所说的 Winsock。网络程序设计者可以非常方便地使用 Winsock 开发基于 TCP 或者 UDP 的应用程序。本章后面要详细讨论这些编程接口。

本书使用节（segment）来表示传输层封包。

1.2.5 网络层（Network Layer）

Internet 的网络层负责将网络层封包从一个主机移动到其他主机，这里的网络封包称为数据报（datagram）。在源主机，Internet 传输层协议（TCP 或 UDP）向网络层传递一个传输层节和一个目的地址，就如同你给邮递员一个带有地址的信。然后，网络层提供将这个节邮递到目的主机传输层的服务。

Internet 的网络层有两个基本组件。一个是 IP 协议，它定义了数据报中各域以及终端系统和路由器如何在这些域上进行操作。仅有一个 IP（Internet Protocol）协议，所有网络层的 Internet 组件都必须运行这个协议。另一个是路由协议，它们用来决定数据报所走的路径。网络层的路由协议很多，因为 Internet 含有多种不同类型的网络，各个网络使用的路由协议有

可能不同。即便是这样，网络层还是经常被人们简单地称为 IP 层，反映了 IP 是将 Internet 绑在一起的胶带。

网络层包含了子网的操作，是懂得网络拓扑结构（网络中机器的物理配置、带宽的限制等）的最高层，也是内网通信的最高层。它的责任是确定数据的物理路径。

1.2.6　链路层（Link Layer）

Internet 的网络层通过一系列的路由器在源地址和目的地址之间传输数据报。为了将封包从路径上的一个节点移动到下一个节点，网络层依赖于链路层的服务。在每个节点，网络层传递数据报到下面的链路层，让它将之发送到路径上的下一个节点。在下一个节点，链路层再把这个数据报传递给网络层。

链路层间的通信方式有两种，一种是将数据发给它所有相邻的节点，这便是广泛用于 LAN（Local Area Network，局域网）的广播通信；另一种是应用于 WAN 中的点对点通信，例如，两个路由器之间或者住宅的拨号调制解调器（Modem）和 ISP 路由之间的通信。对应这两种通信方式的常用协议有 Ethernet 和 Point-to-Point（PPP）。

对一个给定的连接来说，链路层协议主要实现在适配器中，即我们平常所说的 NIC（Network Interface Card，网卡），它有一个主机总线接口和一个连接接口。传输节点的网络层把网络层数据报传递到适配器，由适配器将此数据报封装到链路层的帧中，然后把这个帧传输到物理层通信链路。在另一方，接收适配器接收到整个帧，从中萃取出网络层数据报，将它传给网络层。

本书将使用帧（frame）来表示链路层封包。

1.2.7　物理层（Physical Layer）

链路层的工作是从一个网络节点向其临近的网络节点传送整个帧，其下面的物理层的工作是将帧中的原始比特流从一个节点传送到下一个节点。应用于此层的协议在 TCP/IP 参考模型中并没有定义，它们与连接有关，更依赖于传输介质。例如，以太网有许多物理层协议，有针对双绞线的，有针对同轴电缆的，有针对光纤的，等等。它们都以不同的方式在链接中传送数据位。

1.3　网络程序寻址方式

编写网络程序，必须要有一种机制来标识通信的双方。本节详细讨论 Internet 中各层的寻址方式，以及相关的寻址协议。

1.3.1　MAC 地址

网络通信的最边缘便是 LAN 了，我们先来看看在 LAN 中是如何寻址的。

1．MAC 子层和 MAC 地址

LAN 主要使用广播通信。在其内部，许多主机连在相同的通信通道上，通信时的关键问题是当竞争存在时如何决定谁使用通道。解决此问题的协议属于链路层的子层，称为 MAC

（Medium Access Control，介质访问控制）子层。MAC 子层在 LAN 中特别重要，因为广播通信是由它控制的。

网络中的节点（主机或者路由器）都有链路层地址。事实上，并不是节点有链路层地址，而是节点的适配器有。链路层地址通常叫做 LAN 地址、物理地址或者 MAC 地址（本书统一使用 MAC 地址）。MAC 地址的长度为 6 字节，共有 2^{48} 种可能的取值。这个 6 字节地址通常以十六进制表示，每个字节都用一对十六进制数表示，如 E6-E9-00-17-BB-4B。

适配器在生产时就被永久性地安排了一个 MAC 地址，它记录在适配器的 ROM 中，是不可改变的。另外，MAC 地址空间是由 IEEE 管理的，它保证所有适配器的 MAC 地址都不相同。

2. 局域网通信

当适配器想要发送一个帧到其他适配器时，发送适配器将目的适配器的 MAC 地址插入到封包中，然后以广播的方式将此封包发送到 LAN 中的每一台主机（除了它自己）。每个接收到封包的适配器都会查看包中的目的 MAC 地址是否和自己的 MAC 地址相同，如果相同就萃取出包含的数据报，并将其传递到协议堆栈的上层（网络层），如果不同就直接丢弃。这样一来，只有目的节点的适配器才对接收到的帧进行处理。

有的时候发送适配器想要 LAN 中的所有其他适配器都接收并处理它发送的帧。这种情况下，发送适配器在目的地址域插入一个特定的 MAC 广播地址即可。对使用 6 字节地址的 LAN 来说，广播地址是 48 位全设为 1 的地址，即 FF-FF-FF-FF-FF-FF。

3. 广域网通信

MAC 地址仅应用在 LAN 中，一旦封包从 LAN 的网关出来进入 Internet，链路层地址就不再有用了，这个时候，各路由器是依靠下面所讲的网络层的 IP 地址来寻找目标主机或目标主机所在的 LAN 的。

1.3.2 IP 地址

互联网上的每个主机和路由器都有 IP 地址，它将网络号和主机号编码在一起。此组合是唯一的：原则上，互联网中没有两个机器有相同的 IP 地址。所有的 IP 地址都是 32 位长，在 IP 封包的源地址和目的地址域中使用。要注意，IP 地址指定的并不是主机，而是网络接口（如网卡）。因此，如果一台主机有两个网络接口，它就必须有两个 IP 地址。不过，实际上，大部分主机都只有一个网络接口，也就只有一个 IP 地址。

几十年来，IP 地址都被分成了 5 个类，如图 1.3 所示，这个分配方案称为**分类编址方案**。虽然这种方法现今已经不再使用了，但是在各种文献中还是很常见的。我们待会儿再讨论分类寻址的替换者，即现在使用的分类方法。

类别 A、B、C，分别允许 128 个网络和 16 000 000 个主机、16 384 个网络和 64000 个主机、2 000 000 个网络和 256 个主机。类别 D 用于多播，在这里面，数据报被发送到多个主机。以 1111 开始的 E 类地址保留供今后使用。超过 500 000 个网络现在连接到了 Internet 上，这个数目还在飞快地增加。网络号由非盈利公司 ICANN（Internet Corporation for Assigned Names and Numbers）管理以避免冲突。ICANN 又委派地方权利机关管理部分地址空间，然后再分配给 ISP 和其他公司。

图 1.3　IP 地址格式和分类

网络地址是 32 位的数字，通常以点分十进制的形式写出。在这种格式下，每个 4 字节以十进制形式写出，值为 0～255。例如，32 位的十六进制地址 C0290614 写成十进制为 192.41.6.20。最低的 IP 地址是 0.0.0.0，最高的是 255.255.255.255。

值 0 和-1（即所有位都是 1）有特殊的意义，如图 1.4 所示。0 的意思是本网络和主机，-1 被用作广播地址来指定网络中的所有主机。

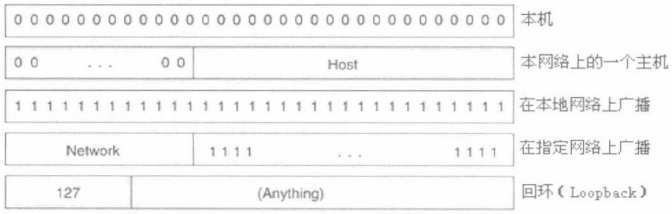


图 1.4　特殊的 IP 地址

IP 地址 0.0.0.0 由主机在引导时使用。网络号为 0 的 IP 地址表示当前网络。这些地址使得网络内的机器在不知道网络号的情况下就可以引用自己所在的网络（但是它们必须要知道它的类，以便知道包含多少个 0）。完全包含 1 的地址允许在本地网络（通常是 LAN）上广播。带有恰当网络号和主机域全为 1 的地址允许机器发送广播包到 Internet 上的任何远程 LAN（不过，大部分网管都禁止这种特性）。最后，所有 127.xx.yy.zz 形式的地址都被保留用作回环测试。发送到这个地址的封包不会被输出到线路上，它们被当作到来的封包直接在本地处理。这允许封包发送到本地网络而发送者不需要知道网络号。

1.3.3　子网寻址

1．子网的概念

使用上述经典分类方法遇到的问题是，单个 A、B 或者 C 类网络地址表示的是一个网络，而不是一组 LAN。为了更有效地利用 IP 地址，人们又将单个网络分成几个部分在内部使用，网络（这里是以太网）中的每个部分称为**子网**（subnet），一个 LAN 就可以是一个子网。

一个网络分成多个子网之后，对外面的世界而言，它仍然是一个单独的网络。典型的校园网网络如图 1.5 所示。它们使用一个主路由器连接 ISP 或者是地方网络，大量以太网分散在校园的不同部门内。每个以太网有自己的路由器，它们连接到主路由器上。

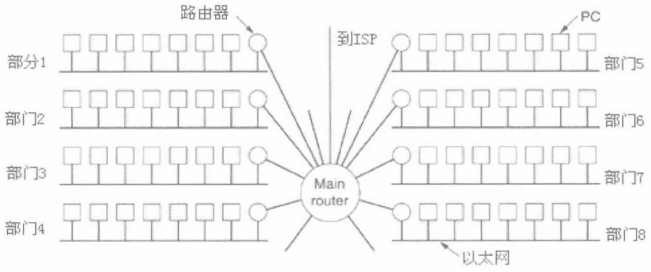


图 1.5 包含各部门 LAN 的校园网

当一个封包到达主路由器时，它如何知道要传给哪个子网呢？一种方法是在主路由器中存放一个包含 6 5536 个入口的表，记录校内的每个主机都使用哪个路由器。这个方法可行，但是它需要在主路由器中存放非常大的表，当主机添加、移除或者终止服务时，要进行许多人工维护。

取而代之的是一种不同的方案。原来单独的 B 类地址中 14 位是网络号，16 位是主机号，但是现在从主机号中拿出几位以创建子网号。例如，如果大学有 35 个部门，它可以拿出 6 位作为子网号，10 位作为主机号，从而允许最多增加 64 个以太网，每个以太网可以最多容纳 1022 个主机。如果错误的话，以后还可以重新划分。

为了实施子网，主路由器需要**子网掩码**，它指定了“网络＋子网＋主机”的各个部分，如图 1.6 所示。子网掩码也以点分十进制形式写出，外加一个斜线，后跟“网络＋子网”部分的位长度。例如在图 1.6 中，子网掩码可以写成 255.255.252.0，也可以写为“/22”，表示子网掩码有 22 位长。

图 1.6 一个划分成了 64 个子网的 B 类网络

在网络外面，子网的划分是不可见的，因此申请一个新的子网不需要惊动 ICANN。在上例中，第一个子网可以使用从 130.50.4.1 开始的 IP 地址，第二个子网从 130.50.8.1 开始，第三个子网从 130.50.12.1 开始，依此类推。为了看到为什么子网间隔数是 4，看看这些地址对应的二进制数据就知道了。

子网 1: 10000010 00110010 000001|00 00000001

子网 2: 10000010 00110010 000010|00 00000001

子网 3: 10000010 00110010 000011|00 00000001

这里，竖直线（|）显示了子网号和主机号之间的边界，它的左边是 6 位的子网号，右边是 10 位的主机号。

2. 子网的工作方式

为了看清楚这些子网是如何工作的，有必要解释一下 IP 封包是如何在路由器中进行处理的。每个路由器有一个表，列出了一些这样的 IP 地址 ——（网络，0）和一些这样的 IP 地

址——（本网络，主机）。第一种说明了封包如何进入远程网络。第二种说明了封包如何到达本地主机。与每个表相关联的是到达目的地要使用的网络接口和其他一些信息。

当 IP 封包到达时，路由器在路由表中查找它的目的地址。如果封包是到远程网络的，它就会在表中记录的接口上被转发到下一个路由器。如果封包是到本地主机的（例如，在路由器的 LAN 上），就会被直接发往目的地。如果表中没有记录，就会被转发到有着更大路由表的默认的路由器上。这种方法意味着，每个路由器仅需要知道其他网络和本地主机，而不是（网络，主机）对，这极大地减小了路由表的大小。

当引入子网划分时，路由表也要改变，添加表的入口——（本网络，子网掩码，0）和（本网络，本子网掩码，主机）。这样，在子网 K 上的路由器便知道如何到达所有其他的子网，也知道如何到达子网 K 上的主机，而不需要知道其他子网上主机的详细信息。事实上，要做的所有改变是使每一个路由器对网络的子网掩码做一个 AND 运算来去掉主机号，然后在表中查找它的地址。例如，一个封包寻址 130.50.15.6，到达了主路由器，使用子网掩码 255.255.252.0/22 做 AND 运算之后，得到地址 130.50.12.0，然后在路由表中查找此地址以便知道使用哪条输出线可以到达 3 号子网。子网的划分就这样通过创建一个包含网络、子网和主机的 3 层结构减小了路由表空间。

1.3.4　端口号

网络层 IP 地址用来寻址指定的计算机或者网络设备，而传输层的**端口号**用来确定运行在目的设备上的哪个应用程序应该接收这个封包。端口号是 16 位的，范围为 0～65 536。在设备上寻址端口号时经常使用的形式是“IP:portnumber”，例如，209.217.52.4:80。连接的两端都要使用端口号，但是没有必要相同。

许多公共服务都使用固定的端口号，例如，WWW（World Wide Web，万维网）默认使用的端口号为 80，FTP（File Transfer Protocol，文件传输协议）使用的是 21，E-mail 使用 25（SMTP，简单邮件传输协议）和 110（POP3，邮局协议）。自定义服务一般使用高于 1 024 的端口号。

1.3.5　网络地址转换（NAT）

IP 地址是短缺的资源。用完 IP 地址的问题并不是会发生在将来的理论问题，它现在就不断地在发生。一个 ISP 可能有一个“/16”地址空间（B 类地址），总共可以有 65 534 个主机号。如果它有更多客户的话，就会出现问题。

对整个 Internet 来说，长期的解决方案便是迁移到 IPv6，它有 128 字节地址。这个转化正在慢慢进行，但是要想真正地完成需要经过很多年的时间。这样，人们就必须找到一个快速的解决办法，能够马上投入使用。这个快速的办法便是 NAT（Network Address Translation，**网络地址转化**），它在 RFC3022 中描述，下面笔者进行概括说明。

NAT 的基本思想是为每个公司分配一个 IP 地址（或者是很少几个）来进行 Internet 传输。在公司内部，每个电脑取得一个唯一的 IP 地址来为内部传输做路由。然而，当封包离开公司，进入 ISP 之后，就需要进行地址转化了。为了使这个方案可行，IP 地址的范围被声明为私有的，公司可以随意在内部使用它们。仅有的规则是，没有包含这些地址的封包出现在 Internet 上。3 个保留的范围是：

10.0.0.0 ～10.255.255.255/8 (16 777 216 台主机)
172.16.0.0 ～172.31.255.255/12 (1 048 576 台主机)
192.168.0.0 ～192.168.255.255/16 (65 536 台主机)

第一个范围提供了 1 677 721 个地址（照常，除了 0 和-1），大部分公司都选择这个范围，即使它们不需要这么多地址。

NAT 操作如图 1.7 所示。在公司内部，每个机器都有一个唯一的 10.x.y.z 形式的地址。然而，当封包离开公司时，它要经过 NAT 盒，此盒将内部 IP 源地址，即图中的 10.0.0.1 转化成公司的真实地址，此例中为 198.60.42.12。NAT 盒通常和防火墙一起绑定在一个设备上，这里的防火墙通过小心地控制进出公司的封包提供了安全保障。本书将在第 12 章讲述防火墙。也可以将 NAT 盒与公司的路由器结合在一起（小的局域网通常是这样）。

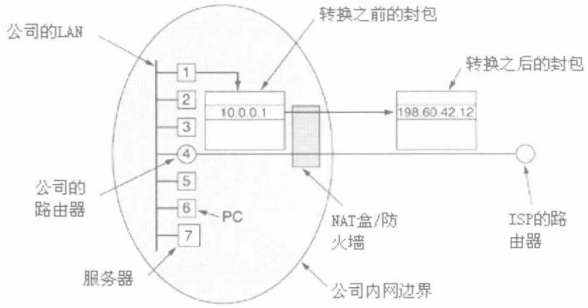


图 1.7 NAT 盒的布置和操作

到此为止，我们忽略了一个很小的细节：当应答返回时（例如，从一个 Web 服务器），它自然是寻址 198.60.42.12，那么，NAT 盒怎样知道该使用哪个地址替换它呢？这是 NAT 要解决的问题。如果在 IP 头中有多余的域，这个域可以用来跟踪真正的发送者是谁，但是现在 IP 头中仅有 1 位还没有使用。原则上，可以创建一个新的 IP 头选项来保存真正的源地址，但是做这件事情需要在整个 Internet 上改变所有机器的 IP 代码以便处理新的选项。作为一个快速解决办法，这实在不怎么样。

真正发生的事情是这样的。NAT 设计者观察到大多数 IP 封包都携带 TCP 或者 UDP 净荷。在第 8 章学习 TCP 和 UDP 时将会看到，它们都有包含源端口号和目的端口号的协议头。端口号是 16 位整型，它指示 TCP 连接从哪里开始和结束。这些端口号提供了使 NAT 工作需要的域。

当进程想和远程进程建立 TCP 连接时，它在自己机器上绑定一个没有使用的 TCP 端口，这称为源端口号，它告诉 TCP 代码向哪里发送到来的封包。这个进程也提供了目的端口号，它说明了在远端将这个封包给谁。端口 0～1024 预留给众所周知的服务。例如，端口 80 是 Web 服务器使用的端口，因此远程客户可以定位它们。每个外出的 TCP 消息都包含目的端口号和源端口号，这些端口号标识了在两个终端使用连接的进程。

使用源端口域能够解决上面的映射问题。每当一个外出的封包进入 NAT 盒，10.x.y.z 源地址被公司的真实地址替换。另外，TCP 源端口号域被一个索引替换，该索引指向 NAT 盒中有 6 5536 个表项的转换表。表中的表项包含了原来的 IP 地址和原来的源端口号。最后，IP 头与 TCP 头的校验和都会被重新计算并插入到封包。替换源端口号是非常必要的，因为

从机器 10.0.0.1 和 10.0.0.2 出发的连接可能恰巧使用了同一个端口，因此，端口号本身不足以标识发送进程。

当封包从 ISP 到达 NAT 盒时，TCP 头中的源端口号被提取出来，用来在 NAT 盒的映射表中当索引。从找到的表项中，内部 IP 地址和原来的 TCP 源端口号被提取出来，并插入到封包。然后，IP 和 TCP 的校验和又重新计算，并插入到封包。最后，封包被传递到公司内部的路由器，使用 10.x.y.z 地址进行正常的发送。

NAT 也可以用来减轻 ADSL 和电缆用户的 IP 短缺。当 ISP 为每个用户分配一个地址时，它使用 10.x.y.z 地址，当来自用户的封包从 ISP 退出，进入主要 Internet 时，它们经由 NAT 盒，NAT 盒将它们转化为真实的 Internet 地址。在回来的路上，封包再经历相反的映射。从这个角度看，对于外部 Internet 来说，ISP 和其 ADLS/电缆用户就像一个大公司。

NAT 确实解决了 IP 地址短缺问题，但是它也带来了一些新的问题。本书后面会看到，NAT 的存在给开发点对点（P2P）应用程序带来了许多麻烦。因为 NAT 设计时并没有考虑让它后面的主机去被动地接受连接，也就是说 NAT 假设它后面的主机不做 Inernet 服务器，所以要想让藏在 NAT 之后的两台主机建立直接的 TCP/UDP 连接，就不得不使用一个中介服务器来帮助它们完成初始化工作。

1.4　网络应用程序设计基础

本节讲述网络应用程序设计的原则和网络程序开发环境的设置。

1.4.1　网络程序体系结构

在创建网络应用程序之前，首先要决定应用程序的体系结构。**应用程序体系结构**（application architecture）由应用程序开发者设计，它指定了在各种各样的终端系统上，应用程序是如何组织的。本节介绍现有的主要体系结构：客户机/服务器体系结构、P2P 体系结构和这两种结构的混合。

1．客户机/服务器体系结构

在**客户机/服务器体系结构**中，有一个总是在运行的主机，称为服务器，它为来自其他许多称为客户的主机提供服务。客户主机可以随时打开和关闭。最通俗的例子就是 Web 应用程序：Web 服务器总是打开的，等待客户端程序（如 IE 浏览器）的请求，通过向它们发送网页数据响应这些请求。客户机/服务器体系结构有如下两个特点：

（1）客户端程序之间并不直接交流信息，它们仅与服务器通信。

（2）服务器方有一个固定的、公开的地址，称为 IP 地址（后面要讨论）。

服务器有固定的地址，而且总是打开的，所以客户端程序才能通过向服务器地址发送封包与之进行通信。

2．P2P（Peer-to-Peer，点对点）体系结构

单纯的 P2P **体系结构**中，不再有总是运行的服务器了，任意的两台主机对（称为 peer）

都可以直接相互通信。因为 peer 之间可以不经过特定的服务器通信，所以这个体系结构称为 peer-to-peer，简写为 P2P。在 P2P 结构中，不再需要任何机器总是打开的，也不再需要任何机器有固定的 IP 地址了。现在，网上有许多著名的 P2P 软件，如疯狂一时的 BT、现今的 eMule、倍受青睐的 QQ 等。

P2P 体系结构的优点之一就是它的可伸缩性。例如，在 P2P 文件共享程序中，数万的 peer 也许会参与到其中，每个 peer 既作为服务器向其他 peer 提供资源，又作为客户端从其他 peer 下载文件。因此，每增加一个 peer，不仅增加了对资源的需求，也增加了对资源的供给。

另一方面，P2P 用户高度分散，它们难以管理。如，有一个重要的文件仅一个 peer 拥有，但是这个 peer 随时都有可能离开网络。

实际上，单纯使用 P2P 体系结构的程序很少，大都需要一个中心服务器来维护总体状态，初始化客户端之间的连接等，这可以算是两种体系结构的混合了。由于网络结构不同，防火墙设置各异，编程时还会遇到更多的问题，如如何穿过内网防火墙、如何穿过 NAT 等，后面会详细介绍。

1.4.2 网络程序通信实体

进程是通信的实体，它们在不同的终端系统上通过计算机网络来交流信息。发送进程创建消息，将之发送到网络，接收进程接收这些消息，发送响应。

1．客户和服务器进程

对于相互通信的两个进程，通常称一方为客户，另一方为服务器。在 Web 里，浏览器是客户进程，Web 服务器是服务器进程。在 P2P 文件共享系统里，下载文件的 peer 称为客户，上传文件的 peer 称为服务器。下面给出客户和服务器进程的具体定义：

在一对进程的通信会话上下文中，初始化通信的进程称为**客户**，等待通信连接的进程称为**服务器**。

2．套接字（Socket）

从一个进程发送到另一个进程的任何消息都必须经过下层网络。进程从网络中接收数据，向网络发送数据都是通过它的**套接字**（Socket）来进行的。为了理解进程和套接字的关系，我们打个比方，进程好比是一个房子，套接字便是房子的门。当进程向其他主机中的进程发送消息时，它将消息推出门（套接字）进入网络。一旦消息到达目标主机，它穿过接收进程的门（套接字），传递给接收进程。所以，套接字便是主机内应用层和传输层的接口，也称为程序和网络间的 API（Application Programming Interface，**应用程序编程接口**）。本书在讲述用户模式网络程序设计时，使用的主要是 Windows 提供的套接字接口。

1.4.3 网络程序开发环境

为了便于直接使用 Windows 提供的网络编程接口，深入了解 Windows 系统网络组件的层次结构，本书主要使用 Visual C++ 6.0 作为编程工具。为了使用 Windows 2000/XP 操作系统的新特性，用户可以更新 SDK 工具。笔者在编写这本书时使用的是 Microsoft Windows Server 2003 SP1 SDK，其下载地址是 http://www.microsoft.com/msdownload/platformsdk/sdkupdate/。

下载 SDK 并安装后，还要对 Visual C++开发环境进行设置。单击菜单“Tools/Options...”，弹出 Options 对话框，选择 Directories 选项卡，首先在“Show directories for:”下拉菜单中选择 Include files，将新 SDK 中头文件的目录添加到“Directories:”列表中，并将其移动到最上方，如图 1.8（左）所示，然后在“Show directories for:”下拉菜单中选择 Library files，进行同样的设置，如图 1.8（右）所示。

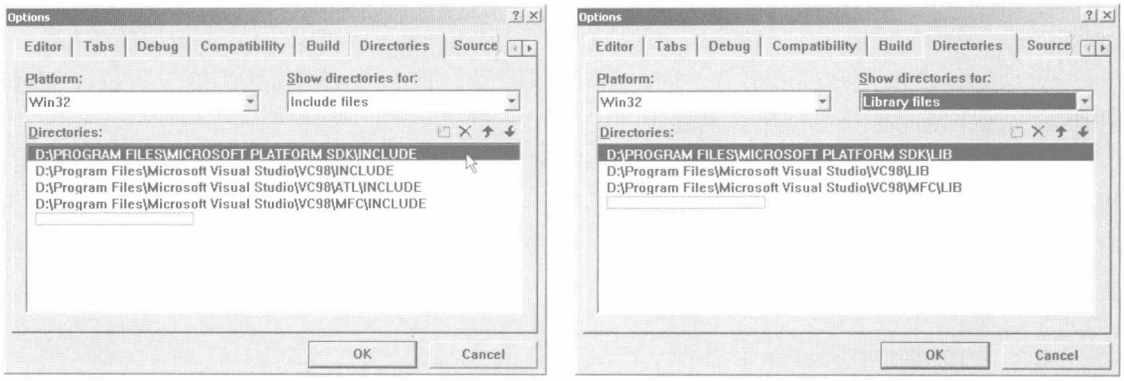


图 1.8　设置 SDK 头文件和库文件目录

在讲述内核网络组件开发时，还需要下载安装 Windows DDK 工具，后面再详细说明。

编写网络程序，调试工具是必不可少的，这里推荐使用免费工具 Dbgview，它可以方便地同时显示内核模式和用户模式下的调试信息。可以在 http://www.sysinternals.com 网站免费下载。

第 2 章　Winsock 编程接口

Winsock 是 Windows 下网络编程的标准接口，它允许两个或多个应用程序在相同机器上，或者是通过网络相互交流。Winsock 是真正的协议无关的接口，本章主要讲述如何使用它来编写应用层的网络应用程序。

2.1　Winsock 库

Winsock 库有两个版本，Winsock1 和 Winsock2。现在开发网络应用程序都使用 Winsock2，需要在程序中包含头文件 winsock2.h，它包含了绝大部分 socket 函数和相关结构类型的声明和定义。同时要添加的还有到 WS2_32.lib 库的链接。包含必要的头文件，设置好链接环境之后，便可进行下面的编码工作了。

2.1.1　Winsock 库的装入和释放

每个 Winsock 应用程序必须加载相应版本的 Winsock DLL。如果在调用 Winsock 函数前没有加载 Winsock 库，函数返回 SOCKET_ERROR，出错代码将是 WSANOTINITIALISED。加载 Winsock 库的函数是 WSAStartup，其定义如下。

```
int WSAStartup(
  WORD wVersionRequested,  // 指定想要加载的 Winsock 库的版本，高字节为次版本号，低字节为主版本号
  LPWSADATA lpWSAData  // 一个指向 WSADATA 结构的指针，用来返回 DLL 库的详细信息
);
```

wVersionRequested 参数用来指定想要加载的 Winsock 库的版本。为了建立此参数的值，可以使用宏 MAKEWORD(x, y)，其中 x 是高字节，y 是低字节。

lpWSAData 是一个指向 LPWSADATA 结构的指针，WSAStartup 使用所加载库的版本信息填充它。

```
typedef struct WSAData {
    WORD            wVersion;                       // 库文件建议应用程序使用的版本
    WORD            wHighVersion;                   // 库文件支持的最高版本
    char            szDescription[WSADESCRIPTION_LEN+1];   // 库描述字符串
    char            szSystemStatus[WSASYS_STATUS_LEN+1];   // 系统状态字符串
    unsigned short  iMaxSockets;                    // 同时支持的最大套接字的数量
    unsigned short  iMaxUdpDg;                      //2.0 版中已废弃的参数
    char FAR *      lpVendorInfo;                   //2.0 版中已废弃的参数
    } WSADATA, FAR * LPWSADATA;
```

函数调用成功返回 0。否则要调用 WSAGetLastError 函数查看出错的原因。此函数的作用相当于 API 函数 GetLastError，它取得最后发生错误的代码。

每一个对 WSAStartup 的调用必须对应一个对 WSACleanup 的调用，这个函数释放 Winsock 库。

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

所有的 Winsock 函数都是从 WS2_32.DLL 导出的，VC++在默认情况下并没有链接到该库，如果想使用 Winsock API，就必须包含相应的库文件。

```
#pragma comment(lib, "WS2_32")
```

2.1.2　封装 CInitSock 类

每次写网络程序都必须编写代码载入和释放 Winsock 库，为了今后讨论方便，这里封装一个 CInitSock 类来管理 Winsock 库，类的使用方法见下一小节。

```
#include <winsock2.h>                          // initsock.h 文件
#pragma comment(lib, "WS2_32")                 // 链接到 WS2_32.lib
class CInitSock
{
public:
    CInitSock(BYTE minorVer = 2, BYTE majorVer = 2)
    {    // 初始化 WS2_32.dll
        WSADATA wsaData;
        WORD sockVersion = MAKEWORD(minorVer, majorVer);
        if(::WSAStartup(sockVersion, &wsaData) != 0)
        {    exit(0);                          }
    }
    ~CInitSock()
    {    ::WSACleanup();              }
};
```

2.2　Winsock 的寻址方式和字节顺序

本节讲述在 Winsock 中主机地址信息的表示方法，以及相关的操作函数。

2.2.1　Winsock 寻址

因为 Winsock 要兼容多个协议，所以必须使用通用的寻址方式。TCP/IP 使用 IP 地址和端口号来指定一个地址，但是其他协议也许采用不同的形式。如果 Winsock 强迫使用特定的寻址方式，添加其他协议就不大可能了。Winsock 的第一个版本使用 sockaddr 结构来解决此问题。

```
struct sockaddr
{
    u_short     sa_family;
    char        sa_data[14];
};
```

在这个结构中，第一个成员 sa_family 指定了这个地址使用的地址家族。sa_data 成员存储的数据在不同的地址家族中可能不同。本书仅仅使用 Internet 地址家族（TCP/IP），Winsock 已经定义了 sockaddr 结构的 TCP/IP 版本——sockaddr_in 结构。它们本质上是相同的结构，但是第 2 个更容易操作。

在 Winsock 中，应用程序通过 SOCKADDR_IN 结构来指定 IP 地址和端口号，定义如下。

```
struct sockaddr_in {
        short    sin_family;                    // 地址家族（即指定地址格式），应为 AF_INET
        u_short sin_port;                       // 端口号
        struct   in_addr sin_addr;              // IP 地址
        char     sin_zero[8];                   // 空字节，要设为 0
};
```

（1）sin_family 域必须设为 AF_INET，它告诉 Winsock 程序使用的是 IP 地址家族。

（2）sin_port 域指定了 TCP 或 UDP 通信服务的端口号。应用程序在选择端口号时必须小心，因为有一些端口号是保留给公共服务使用的，如 FTP 和 HTTP。基本上，端口号可分成如下 3 个范围：公共的、注册的、动态的（或私有的）。

- 0～1 023 由 IANA（Internet Assigned Numbers Authority）管理，保留为公共的服务使用。
- 1 024～49 151 是普通用户注册的端口号，由 IANA 列出。
- 49 152～65 535 是动态和/或私有的端口号。

普通用户应用程序应该选择 1 024～49 151 的注册了的端口号，以避免使用了一个其他应用程序或者系统服务已经使用的端口号。在 49 152～65 535 之间的端口号也可以自由地使用，因为没有服务注册这些端口号。

（3）sin_addr 域用来存储 IP 地址（32 位），它被定义为一个联合来处理整个 32 位的值，两个 16 位部分或者每个字节单独分开。描述 32 位 IP 地址的 in_addr 结构定义如下。

```
struct in_addr {
        union {
                struct { u_char s_b1,s_b2,s_b3,s_b4; } S_un_b;      // 以 4 个 u_char 来描述
                struct { u_short s_w1,s_w2; } S_un_w;               // 以 2 个 u_short 来描述
                u_long S_addr;                                      // 以 1 个 u_long 来描述
        } S_un;
    };
```

用字符串“aa.bb.cc.dd”表示 IP 地址时，字符串中由点分开的 4 个域是以字符串的形式对 in_addr 结构中的 4 个 u_char 值的描述。由于每个字节的数值范围是 0~255，所以各域的值都不可超过 255。

（4）最后一个域 sin_zero 没有使用，是为了与 SOCKADDR 结构大小相同才设置的。

应用程序可以使用 inet_addr 函数将一个由小数点分隔的十进制 IP 地址字符串转化成由 32 位二进制数表示的 IP 地址。inet_ntoa 是 inet_addr 函数的逆函数，它将一个网络字节顺序的 32 位 IP 地址转化成字符串。

```
unsigned long inet_addr(const char* cp);      // 将一个"aa.bb.cc.dd"类型的 IP 地址字符串转化为 32 位的二进制数
char * inet_ntoa (struct in_addr in);         // 将 32 位的二进制数转化为字符串
```

注意，inet_addr 返回的 32 位二进制数是用网络顺序存储的，下一小节详细讲述字节顺序。

2.2.2 字节顺序

字节顺序是长度跨越多个字节的数据被存储的顺序。例如，一个 32 位的长整型 0x12345678 跨越 4 个字节（每个字节 8 位）。Intel x86 机器使用**小尾顺序**（little-endian），意思是最不重要的字节首先存储。因此，数据 0x12345678 在内存中的存放顺序是 0x78、0x56、0x34、0x12。大多数不使用小尾顺序的机器使用**大尾顺序**（big-endian），即最重要的字节首

先存储。同样的值在内存中的存放顺序将是 0x12、0x34、0x56、0x78。因为协议数据要在这些机器间传输，所以就必须选定其中的一种方式做为标准，否则会引起混淆。

TCP/IP 统一规定使用大尾方式传输数据，也称为网络字节顺序。例如，端口号（它是一个 16 位的数字）12345（0x3039）的存储顺序是 0x30、0x39。32 位的 IP 地址也是以这种方式存储的，IP 地址的 4 部分存储在 4 个字节中，第一部分存储在第一个字节中。

上述 sockaddr 和 sockaddr_in 结构中，除了 sin_family 成员（它不是协议的一部分）外，其他所有值必须以网络字节顺序存储。Winsock 提供了一些函数来处理本地机器的字节顺序和网络字节顺序的转换。

```
u_short htons(u_short hostshort);        // 将 u_short 类型变量从主机字节顺序转化到 TCP/IP 网络字节顺序
u_long htonl(u_long hostlong);           // 将 u_long 类型变量从主机字节顺序转化到 TCP/IP 网络字节顺序
u_short ntohs(u_short netshort);         // 将 u_short 类型变量从 TCP/IP 网络字节顺序转化到主机字节顺序
u_long ntohl(u_long netlong);            // 将 u_long 类型变量从 TCP/IP 网络字节顺序转化到主机字节顺序
```

这些 API 是平台无关的。使用它们可以保证程序正确地运行在所有机器上。

下面代码示例了如何初始化 sockaddr_in 结构。

```
sockaddr_in sockAddr;
// 设置地址家族
sockAddr.sin_family = AF_INET;
// 转化端口号 6789 到网络字节顺序，并安排它到正确的成员
sockAddr.sin_port = htons(6789);
// inet_addr 函数转化一个"aa.bb.cc.dd"类型的 IP 地址字符串到长整型
// 它是以网络字节顺序记录的 IP 地址
sockAddr.sin_addr.S_un.S_addr = inet_addr("127.0.0.1");
// 也可以用下面的代码设置 IP 地址（通过设置 4 个字节部分，设置 sockAddr 的地址）
/*  sockAddr.sin_addr.S_un.S_un_b.s_b1 = 127;
    sockAddr.sin_addr.S_un.S_un_b.s_b2 = 0;
    sockAddr.sin_addr.S_un.S_un_b.s_b3 = 0;
    sockAddr.sin_addr.S_un.S_un_b.s_b4 = 1;      */
```

2.2.3　获取地址信息

通常，主机上的接口被静态地指定一个 IP 地址，或者是由配置协议来分配，如动态主机配置协议（DHCP）。如果 DHCP 服务器不能到达，系统会使用 Automatic Private IP Addressing（APIPA）自动分配 169.254.0.0/16 范围内的地址。

1．获取本机 IP 地址

获取本机的 IP 地址比较简单，下面的 GetAllIps 例子打印出了本机使用的所有 IP（一个适配器一个 IP 地址），程序代码如下。

```
#include "../common/InitSock.h"                                    // GetAllIps 工程
#include <stdio.h>
CInitSock initSock;                    // 初始化 Winsock 库
void main()
{
    char szHost[256];
    // 取得本地主机名称
    ::gethostname(szHost, 256);
    // 通过主机名得到地址信息
```