图 12-1　TCP/IP 协议栈的四个层次

每一层负责不同的功能，下面分别介绍。

（1）链路层，有时也称作数据链路层或网络接口层，通常包括操作系统中的设备驱动程序和计算机中对应的网络接口卡。它们一起处理与电缆（或其他任何传输媒介）的物理接口细节。

（2）网络层，有时也称作互联网层，处理分组在网络中的活动，例如分组的路由选择。在 TCP/IP 协议组件中，网络层协议包括 IP 协议（网际协议），ICMP 协议（Internet 互联网控制报文协议），以及 IGMP 协议（Internet 组管理协议）。

（3）传输层主要为两台主机上的应用程序提供端到端的通信。在 TCP/IP 协议组件中，有两个互不相同的传输协议：TCP（传输控制协议）和 UDP（用户数据报协议）。

TCP 为两台主机提供高可靠性的数据通信。它所做的工作包括把应用程序交给它的数据分成合适的小块交给下面的网络层，确认接收到的分组，设置发送最后确认分组的超时时钟等。由于运输层提供了高可靠性的端到端的通信，因此应用层可以忽略所有这些细节。

而另一方面，UDP 则为应用层提供一种非常简单的服务。它只是把称作数据报的分组从一台主机发送到另一台主机，但并不保证该数据报能到达另一端。任何必需的可靠性必须由应用层来提供。

这两种运输层协议分别在不同的应用程序中有不同的用途，这一点将在后面介绍。

（4）应用层负责处理特定的应用程序细节。几乎各种不同的 TCP/IP 实现都会提供下面这些通用的应用程序（其实也是根据相关的协议写成）：

- Telnet 远程登录
- FTP 文件传输协议
- SMTP 用于电子邮件的简单邮件传输协议

• SNMP 简单网络管理协议

假设我们在一个局域网（LAN）如以太网中有两台主机，二者都运行 FTP 协议，图 12-2 列出了该过程所涉及到的所有协议。

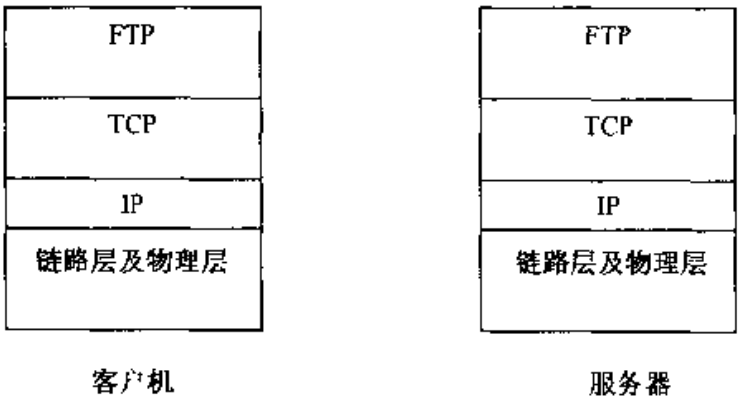

图 12-2 局域网上运行 FTP 的两台主机

这里，我们列举了一个 FTP 客户程序和另一个 FTP 服务器程序。大多数的网络应用程序都被设计成客户/服务器模式。服务器为客户提供某种服务，在本例中就是访问服务器所在主机上的文件。在远程登录应用程序 Telnet 中，为客户提供的服务是登录到服务器主机上。

在同一层上，双方都有对应的一个或多个协议进行通信。例如，某个协议允许 TCP 层进行通信，而另一个协议则允许两个 IP 层进行通信。

在图 12-2 的右边，我们注意到应用程序通常是一个用户进程，而下三层则一般在（操作系统）内核中执行。尽管这不是必需的，但通常都是这样处理的，例如 UNIX 操作系统。

在图 12-2 中，顶层与下三层之间还有另一个关键的不同之处。应用层关心的是应用程序的细节，而不是数据在网络中的传输活动。下三层对应用程序一无所知，它们要处理所有的通信细节。

我们在图 12-2 中列举了四种不同层次上的协议。FTP 是一种应用层协议，TCP 是一种运输层协议，IP 是一种网络层协议，而以太网协议则应用于链路层上。TCP/IP 协议组件是一组不同的协议组合在一起构成的协议族。尽管通常称该协议组件为 TCP/IP，但 TCP 和 IP 只是其中的两种协议而已（该协议组件的另一个名字是 Internet 协议族(Internet Protocol Suite)。

12.2.2 TCP 和 UDP 协议

TCP 和 UDP 是两种最为著名的运输层协议，二者都使用 IP 作为网络层协议。虽然 TCP 使用不可靠的 IP 服务，但它却提供一种可靠的运输层服务。UDP

为应用程序发送和接收数据报。一个数据报是指从发送方传输到接收方的一个信息单元（例如，发送方指定的一定字节数的信息）。但是与 TCP 不同的是，UDP 是不可靠的，它不能保证数据报能安全无误地到达最终目的。

12.2.3　IP 协议和和 IP 地址

IP 是网络层上的主要协议，同时被 TCP 和 UDP 使用。TCP 和 UDP 的每组数据都通过端系统和每个中间路由器中的 IP 层在互联网中进行传输。

互联网上的每个接口必须有一个惟一的 Internet 地址（也称作 IP 地址）。IP 地址长 32bit。Internet 地址并不采用平面形式的地址空间，如 1，2，3 等。IP 地址具有一定的结构，共有五类不同的互联网地址格式。

这些 32 位的地址通常写成四个十进制的数，其中每个整数对应一个字节。这种表示方法称作“点分十进制表示法”（dotted decimal notation）。例如，本人机器上的 IP 地址就是一个 C 类地址，它表示为：202.112.113.114。

区分各类地址的最简单方法是看它的第一个十进制整数。它可以写成二进制形式，A 类地址的格式是 0XXXXXXX.XXXXXXXX.XXXXXXXX.XXXXXXXX；相应的 B 类地址的格式是 10XXXXXX.XXXXXXXX.XXXXXXXX.XXXXXXXX；C 类地址格式为：110XXXXX.XXXXXXXX.XXXXXXXX.XXXXXXXX；D 类地址格式为：1110XXXX.XXXXXXXX.XXXXXXXX.XXXXXXXX；E 类地址格式为：11110XXX.XXXXXXXX.XXXXXXXX.XXXXXXXX，等等。

需要再次指出的是，多接口主机具有多个 IP 地址，其中每个接口都对应一个 IP 地址。

由于互联网上的每个接口必须有一个惟一的 IP 地址，因此必须要有一个管理机构为接入互联网的网络分配 IP 地址。这个管理机构就是互联网络信息中心（Internet Network Information Centre）称作 InterNIC。InterNIC 只分配网络号，主机号的分配由系统管理员来负责。

12.2.4　域名系统

尽管通过 IP 地址可以识别主机上的网络接口，进而访问主机，但是人们最喜欢使用的还是主机名。在 TCP/IP 领域中，域名系统（DNS）是一个分布的数据库，由它来提供 IP 地址和主机名之间的映射信息。

现在，我们必须理解，任何应用程序都可以调用一个标准的库函数来查看给定名字的主机 IP 地址。类似地，系统还提供一个逆函数——给定主机的 IP 地址，查看它所对应的主机名。

大多数使用主机名作为参数的应用程序也可以把 IP 地址作为参数。例如，

当我们用 Telnet 进行远程登录时，既可以指定一个主机名，也可以指定一个 IP 地址。

12.2.5 封装

当应用程序用 TCP 传送数据时，数据被送入协议栈中，然后逐个通过每一层直到被当作一串比特流送入网络。其中每一层对收到的数据都要增加一些首部信息（有时还要增加尾部信息）。TCP 传给 IP 的数据单元称作 TCP 报文段或简称为 TCP 段（TCP Segment）。IP 传给网络接口层的数据单元称作 IP 数据包(IP Datagram)。通过以太网传输的比特流称作帧(Frame)。

帧头和帧尾下面所标注的数字是典型以太网帧首部的字节长度。以太网数据帧的物理特性是其长度必须在 46~1500 字节之间。

UDP 数据与 TCP 数据基本一致。惟一不同的是 UDP 传给 IP 的信息单元称作 UDP 数据报（UDP datagram），而且 UDP 的首部长为 8 字节。

12.2.6 解析

当目的主机收到一个以太网数据帧时，数据就开始从协议栈中由底向上传递，同时去掉各层协议加上的报文首部。每层协议盒都要去检查报文首部中的协议标识，以确定接收数据的上层协议。这个过程称作解析。

12.2.7 客户服务器模型

大部分网络应用程序在编写时都假设一端是客户，另一端是服务器，其目的是为了让服务器为客户提供一些特定的服务。

我们可以将这种服务分为两种类型：重复型或并发型。重复型服务器通过以下步骤进行交互：

I1：等待一个客户请求的到来；

I2：处理客户请求；

I3：发送响应给发送请求的客户；

I4：返回 I1 步骤。

重复型服务器主要的问题发生在 I2 状态。在这个时候，它不能为其他客户机提供服务。

相应地，并发型服务器采用以下步骤：

C1：等待一个客户请求的到来。

C2：启动一个新的服务器来处理这个客户的请求。在这期间可能生成一个新的进程、任务或线程，并依赖底层操作系统的支持。这个步骤如何进行取决于

操作系统。生成的新服务器对客户的全部请求进行处理。处理结束后，终止这个新服务器。

C3：返回 C1 步骤。

并发服务器的优点在于它是利用生成其他服务器的方法来处理客户的请求。也就是说，每个客户都有它自己对应的服务器。如果操作系统允许多任务，那么就可以同时为多个客户同时服务。

我们对服务器，而不是对客户进行分类的原因是因为对于一个客户来说，它通常并不能够辨别自己是在与一个重复型服务器还是并发型服务器进行对话。

一般来说，TCP 服务器是并发的，而 UDP 服务器是重复的，但也存在一些例外。

12.2.8　端口号

TCP 和 UDP 采用 16 比特的端口号来识别应用程序。那么这些端口号是如何选择的呢？

服务器一般都是通过人们所熟知的端口号来识别的。例如，对于每个 TCP/IP 实现来说，FTP 服务器的 TCP 端口号都是 21，每个 Telnet 服务器的 TCP 端口号都是 23，每个 TFTP(简单文件传输协议)服务器的 UDP 端口号都是 69。任何 TCP/IP 实现所提供的服务都用众所周知的 1~1023 之间的端口号。这些人们所熟知的端口号由 Internet 端口号分配机构（Internet Assigned Numbers Authority, IANA）来管理。

客户端通常对它所使用的端口号并不关心，只需保证该端口号在本机上是惟一的就可以了。客户端口号又称作临时端口号（即存在时间很短暂），这是因为它通常只是在用户运行该客户程序时才存在，而服务器则只要主机开着的，其服务就运行。

大多数 TCP/IP 实现给临时端口分配 1024~5000 之间的端口号。大于 5000 的端口号是为其他服务器预留的（Internet 上并不常用的服务）。

12.2.9　应用编程接口

使用 TCP/IP 协议的应用程序通常采用两种应用编程接口（API）：Socket 和 TLI（运输层接口：Transport Layer Interface）。前者有时称作“Berkeley socket”，表明它是从伯克利版发展而来的。后者起初是由 AT&T 开发的，有时称作 XTI（X/Open 传输接口），以承认 X/Open 这个自己定义标准的国际计算机生产商所做的工作。XTI 实际上是 TLI 的一个超集。在下一章我们将主要介绍 socket 编程机制。我们后面所介绍的网络编程的例子，就是用 socket 来实现的。

小　结

本章主要介绍网络基本原理，尤其是涉及 Internet 的基本结构和原理，TCP/IP 协议及其应用等。

第十三章 WinSock 编程机制

Socket 原来是 UNIX 的 Berkeley Software Distribution 版本中的一个程序接口。它类似于 C 的函数库。简单地说，WinSock 是定义于视窗应用程序（Windows Application）与网络（Network）之间的标准界面。程序设计者在发展视窗的网络应用软件时，只要支持 WinSock 界面的标准规格，就不再需要顾虑所使用的网卡硬件部分，因为 WinSock 所提供的动态链接程序库（WS2_32.DLL）会完成网络底层沟通的工作。进而使得程序设计者能设计出更多功能或更友善的网络软件。下面我们将具体介绍 WinSock 的编程框架和具体函数。

13.1 WinSock 描述字

在多数的操作系统中，当应用程序需要执行 I/O 操作，要求操作系统打开一个文件时，系统常常建立一个文件描述字，然后应用程序利用这个文件描述字操作相应的文件。文件描述字是整型，它表示该文件状态信息在系统表的索引。Socket 与文件类似，每一个活动的 Socket 由一个整数标识，这个整数被称为 Socket 描述字。Windows 操作系统会为 Socket 重新安排一个新的系统表。这样文件描述字和 Socket 描述字可能具有相同的值。

Socket API 中包含一个函数 Socket()，应用程序通过它来建立 Socket，它的返回值就是 Socket 描述字。一旦 Socket 建立了，服务器建立的 Socket 被用来等待请求，我们称其为消极 Socket。客户机建立的 Socket 被用来发起连接请求，我们称其为积极 Socket。

13.2 Socket 的系统数据结构

当应用程序调用 Socket 函数时，操作系统分配一个新的数据结构来存储通讯所需要的状态信息，同时将这个存放状态信息的数据指针存放在进程 Socket 描述表中。下图描述了 Socket 的内部数据结构。Family 指协议族，PF_INET 是 INTERNET 协议族的意思，SOCK_STREAM 是面向流的服务。

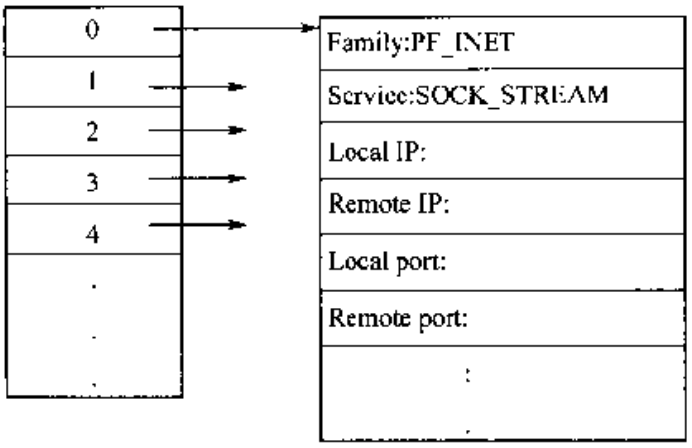

图 13-1 Socket 的系统数据结构

13.3 Socket 的地址说明

当一个 Socket 建立后，它并不包括任何关于它将如何被使用的细节信息。特别地，它不包含关于使用它的本地或远程机器的地址信息。因而在应用程序使用 Socket 时，它必须说明这两个地址。TCP/IP 定义它的端点信息由 IP 地址或协议端口号组成，当然其他的协议族有不同的定义方法。在本书中，我们主要涉及到 TCP/IP 协议族，对于其他的我们不作介绍了。TCP/IP 使用的地址族是由 AF_INET 标识的。

13.4 Socket API 的函数说明

编写网络程序的方法很多，可以直接使用一些开发包，但是要编写基本的网络程序，就需要使用 Socket API 函数。了解 Socket API 函数的好处是能清楚了解 Socket 中发送和接收数据的机制和过程。所有的其他开发包都是构建在 Socket API 基础之上的。这里只介绍常用的几个 API 函数。

1. WSAStartup 函数

进程在每次使用 Windows 套接字必须使用 WSAStartup 函数来初始化 WS2_32.DLL 动态连接库，注意在每次调用 WSAStartup 函数并完成相应的网络功能后要调用 WSACleanup 函数。

```
int WSAStartup (
  WORD wVersionRequested,
```

```
    LPWSADATA lpWSAData
);
```

表 13-1 参数说明

参 数	说 明
WversionRequested	用于指定加载的套接字的版本，高位字节用来指定套接字的次版本，低位用来指定套接字的主版本号，可以使用 MAKEWORD(X, Y) （其中，x 是高位字节，y 是低位字节）方便的获得 wVersionRequested 的正确值
LpWSAData	指向 LPWSADATA 结构的指针，WSAStartup 函数用其加载的库版本的信息填充该结构
输出参数	成功调用则返回 0，否则非 0 注意：不能使用 WSAGetLastError 函数来判定发生错误的类型，因为此时，套接字的动态连接库并没有加载，因此 WSAGetLastError 函数所使用的客户数据区的错误信息并没有建立

事实上，在用户建立 Socket 应用程序时，应用程序与 Socket 套接字库有一个协商的过程。应用程序和 WS2_32.DLL 动态链接库的协商过程如表 13-1 所示。

表 13-2 应用程序和动态链接库的协商过程

应用程序指定版本	DLL 版本	Wversion Requested	wVersion	wHigh Version	最终结果
1.1	1.1	1.1	1.1	1.1	使用 1.1
1.0 1.1	1.0	1.1	1.0	1.0	使用 1.0
1.0	1.0 1.1	1.0	1.0	1.1	使用 1.0
1.1	1.0 1.1	1.1	1.1	1.1	使用 1.1
1.1	1.0	1.1	1.0	1.0	应用程序错
1.0	1.1	1.0	---	---	版本不支持
1.0 1.1	1.0 1.1	1.1	1.1	1.1	使用 1.1
1.1 2.0	1.1	2.0	1.1	1.1	使用 1.1
2.0	2.0	2.0	2.0	2.0	使用 2.0

下面代码示范如何使用 WSAStartup 函数：

```
WORD wVersionRequested;
WSADATA wsaData;
int err;
```

```
wVersionRequested = MAKEWORD( 2, 2 );

err = WSAStartup( wVersionRequested, &wsaData );
if ( err != 0 ) {
    /* 无法找到一个可用的 WinSock DLL.*/
    return;
}

/* 判定套接字动态链接库 WinSock DLL 是否支持 2.2 版本.*/
/* 注意如果动态链接库版本既可支持 2.2，又高于 2.2   */
/* 此时它会返回 2.2，因为应用程序请求的版本是 2.2   */

if ( LOBYTE( wsaData.wVersion ) != 2 ||
        HIBYTE( wsaData.wVersion ) != 2 ) {
    /* 用户应寻找一个合适的套接字动态链接库 WinSock DLL */
    WSACleanup( );
    return;
}

/* 继续其他套接字函数调用 */
```

2. WSACleanUp 函数

当应用程序完成网络功能，不再使用和准备关闭 Socket 时，应用程序调用 WSACleanup 函数，结束对 WS2_32.DLL 动态链接库的使用。

表 13-3 参数说明

参　数	说　明
输出参数	成功调用则返回 0，否则返回 SOCKET_ERROR。这时你可以使用 WSAGetLastError 函数来判断发生错误的类型

WSAGetLastError 返回的错误代码如下：

WSANOTINITIALISED：　没有初始化 WINSOCKET

WSAENETDOWN：　网络系统错

WSAEINPROGRESS：　一个 WINSOCKET1.1 版本的阻塞调用正在处理或者服务提供者正在处理一个回调函数