鲁斌 主编
李莉 朵春红 副主编

网络程序开发实用案例教程

清华大学出版社

网络程序开发
实用案例教程

鲁斌 主编
李莉 朵春红 副主编

清华大学出版社
北京

内 容 简 介

本书是一本综合、全面、实用的网络编程实验教材，书中精选了12个网络编程典型案例，基本涵盖了当前网络编程领域的热点问题，内容包括网络程序开发基础、FTP服务器与客户端、Web服务器、电子邮件系统、网络防火墙、网络流量监控系统、网站下载程序、网络五子棋、语音聊天系统、远程控制系统、办公自动化系统、基于B/S的即时通信系统以及通用课程教学网站等。通过本书的学习，能够使读者掌握目前最流行的Windows C/S模式和B/S模式的网络应用程序的开发技术，从而胜任任何复杂程序的设计与开发要求。本书可用作高等学校网络工程及其相关专业高年级本科生和研究生的实验教材或教学参考书，也可供其他技术开发人员参考。

图书在版编目（CIP）数据

网络程序开发实用案例教程/鲁斌主编. —北京：清华大学出版社，2011.10
ISBN 978-7-302-24403-5

Ⅰ. ①网… Ⅱ. ①鲁… Ⅲ. ①主页制作—程序设计—高等学校—教材 Ⅳ. ①TP393.092

中国版本图书馆CIP数据核字(2010)第260368号

责任编辑：汪汉友
责任校对：梁　毅
责任印制：何　芊

出版发行：清华大学出版社　　**地　　址**：北京清华大学学研大厦A座
http://www.tup.com.cn　　**邮　　编**：100084
社　总　机：010-62770175　　**邮　购**：010-62786544
投稿与读者服务：010-62795954，jsjjc@tup.tsinghua.edu.cn
质　量　反　馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
印 刷 者：北京市清华园胶印厂
装 订 者：三河市金元印装有限公司
经　　销：全国新华书店
开　　本：185×260　　**印　张**：22.5　　**字　数**：559千字
版　　次：2011年10月第1版　　**印　次**：2011年10月第1次印刷
印　　数：1～3000
定　　价：35.00元

产品编号：036684-01

前　言

计算机网络技术的发展突飞猛进，日新月异，对网络程序开发人员的程序设计水平和动手实践能力提出了很高的要求。网络编程在程序设计领域所处的地位变得越来越重要，是对网络知识和软件知识的综合应用，体现了软件技术发展的热门方向。

当前，尽管已出版的网络编程书籍不在少数，但大多介绍的是基本的网络程序设计原理和技术以及部分网络编程内容。对于更高级、更全面、更系统的网络应用，还没有简单的途径进行学习、设计与开发。

本书正是针对上述问题而撰写的。这是一部综合、全面、实用的网络编程实验教材，根据作者多年来的教学思路和经验，精选了 12 个网络编程典型案例，基本上涵盖了当前网络编程领域的热点问题。与已出版的同类书籍相比，本书具有内容全面、重点突出、层次分明等特点，具体体现在：

(1) 内容全面。本书所选的网络编程案例内容涵盖了各种主要的网络程序设计技术，从基于 Winsock 的网络编程技术到基于 ASP. NET 的 Web 编程技术，应有尽有。

(2) 重点突出。网络编程技术和工具种类繁多，不可能面面俱到。本书精心选取了目前主流的网络编程案例进行深入而透彻的分析，从而使得程序员完全可以胜任任何复杂程序设计任务的开发要求。

(3) 层次分明。本书是对作者长期教学与实践经验的良好总结，按照层次化的教学理念对知识点的逻辑结构进行组织。讲解每个案例时，首先介绍案例涉及的基本知识、原理和技术，然后分析案例的设计思路和过程，最后给出具体的代码实现方案，循序渐进，按照学习规律和对事物认知的过程安排章节内容。

本书全部实例程序源代码均给出了详细的中文注释和说明，并且能够不做任何修改即可编译运行，极大地方便了读者学习和使用。读完本书并且亲自动手实践随书程序案例后，读者将能够系统而全面地掌握面向 C/S 架构和 B/S 架构的各种主要的网络编程技术。

本书在内容的安排上，首先简洁明了地介绍了网络编程所涉及的基本知识，接着分两个部分详细介绍了这些典型案例的设计与实现过程。第一部分涵盖第 2 章至第 10 章，系统地介绍了基于 Windows Socket 编程接口的各种网络编程典型案例，内容包括 FTP 服务器与客户端、Web 服务器、电子邮件系统、网络防火墙、网络流量监控系统、网站下载程序、网络五子棋、语音聊天系统以及远程控制系统等；第二部分涵盖第 11 章至第 13 章，以 ASP. NET 为基础介绍了 Web 编程典型案例，内容包括办公自动化系统、基于 B/S 的即时通信系统以及通用课程教学网站等。上述内容安排能够有效地指导读者如何利用现有的主流开发技术进行最流行的 Windows C/S 模式和 B/S 模式的网络应用程序的开发技术。

本书由鲁斌主编，并负责第 1 章～第 3 章、第 5 章、第 6 章、第 9 章、第 10 章的撰写工作；第 4 章、第 7 章、第 8 章由朵春红编写，最后 3 章由李莉编写，鲁斌负责统稿。同时，特别

感谢王翠茹教授、胡朝举副教授审阅了全书并对本书提出了宝贵建议。另外，陈娟、宋亚奇、张强、王艳丽和刘换霞在本书创作之初做了大量的素材搜集工作，在此一并表示感谢。

由于编者水平有限，错误之处在所难免，恳请广大读者批评指正。

本书可用作高等学校网络工程及其相关专业高年级本科生和研究生的实验教材或教学参考书，也可供其他技术开发人员参考。

鲁　斌

2011 年 5 月

目　录

第 1 章　网络程序开发基础

套接字(Socket)编程接口是迄今为止最为常用、最为重要的一类网络编程接口，目的是解决网间进程通信的问题。Socket 的英文原意是“孔”或“插座”，它在进程通信机制中起的作用的确就像插座一样。当 Socket 接通时，用户可以通过它来接收对方发来的任何信息，也可以将文件传输到网络中的任何地方，只要对方在线，且对方的 Socket 同自己的 Socket 有通信连接，这两者之间就可以实现通信。

最早的套接字规范源自 Berkeley 大学，这个规范是针对 UNIX 操作系统下的 TCP/IP 协议实现的，为在 UNIX 操作系统下不同计算机之间使用 TCP/IP 协议进行网络通信编程提供了一套 API，使得程序员在编写网络应用程序时只需调用这些函数，对网络的底层细节并不要求精通而只需熟悉即可，因而给编程人员带来了很大的便利。由于 Berkeley 大学最先涉及 Socket 接口开发工作，因此这个套接字规范一般称为 Berkeley Socket。

随着个人计算机的日益普及，Windows 操作系统的用户与日俱增。为了使得原先在 UNIX 上才能实现便捷的网络通信方式同样也能在 Windows 上得以实现，Microsoft 以 Berkeley Socket 规范为范例定义了一套 Windows 下的网络编程接口，即 Windows Socket 编程接口，简称 Winsock。它不仅包含了人们很熟悉的 Berkeley Socket 风格的库函数，也包含了一组针对 Windows 的扩展库函数，以使程序员能充分利用 Windows 消息驱动机制进行编程。并且，Winsock 是一份独立的规范，它的产生和存在是为了适应应用程序开发者、网络软件供应商和广大计算机用户的需求。这份规范每一次正式出版的版本实际上都代表了网络软件供应商实现所需和应用开发者所用的一整套 API。从 1991 年的 1.0 版到 1997 年的 2.2.1 版，经过不断完善并在 Intel、Microsoft、Sun、SGI 等公司的全力支持下，Winsock 已成为 Windows 网络编程事实上的标准。

1.1　套接字的类型

套接字是通信的基础，是支持网络协议数据通信的基本接口。从某种意义上讲，可以将套接字看作是不同主机的进程进行网络通信的端点，它构成了应用程序与整个网络间的编程界面。套接字存在于通信域中，且一般与同一个域中的套接字交换数据。Winsock 支持单一的通信域，各种进程都使用这个单一域即 Internet 域来进行通信；与之不同，Winsock 2 范围要广一些，支持多种通信域。

套接字的类型与网络协议密切相关。套接字类型不同，所对应的网络协议也不同。常见的网络协议有两种：一种是用户数据报协议(User Datagram Protocol，UDP)，这是一种面向消息、无连接的、不可靠但效率很高的协议；另一种是传输控制协议(Transfer Control Protocol，TCP)，是一种基于字节流的、面向连接的、可靠的协议。网络编程时，要根据应用的需求，选择适当的协议，对于要求大量可靠的数据传输，应当选择 TCP 协议，否则选择 UDP 协议。

套接字根据通信性质可以分为流式套接字、数据报套接字以及原始套接字 3 种。

1.1.1 流式套接字

流式套接字是 TCP 协议的重要体现，它提供了双向的、有序的、无重复以及无记录边界的数据流服务，它非常适合于处理大量数据。

流式套接字是面向连接的，即在进行数据交换之前，要先建立数据传输链路，这样就为后继数据的传输确定了可以确保有序到达的路径，同时为了确保数据的正确性，可能还会执行额外的计算来验证正确性，所以相对于数据报套接字，它的系统开销较大。

1.1.2 数据报套接字

数据报套接字是 UDP 协议的重要体现，它支持双向的数据流，但在传输过程中并不保证数据的传输可靠性、有序性和无重复性。任何数据一旦被发出，都不能保证该数据能够完好无损、正确无误地被对方接收。除此之外，数据报套接字还有一个重要特点，就是它保留了记录边界。

由于数据报套接字是无连接的，所以在数据传输时，它并不能保证接收端是否正在侦听。因此数据报并不十分可靠，需要编码来实现数据的排序和传输的可靠性，但由于它的传输效率非常高，所以至今还得到较为广泛的应用。

1.1.3 原始套接字

原始套接字允许直接访问较低层次的协议（如 IP、ICMP），用于检验新协议的实现。Winsock 规范并没有规定 Winsock DLL 必须支持这种套接字，然而，Winsock 规范鼓励 Winsock DLL 提供对原始套接字的支持。

1.2 Winsock 编程的一般模式

Winsock 编程一般基于客户机/服务器（Client/Server，C/S）模型实现，并非像想象中那样复杂，它要遵循一些基本而且必要的编程模式或者说是步骤的，下面将分类进行介绍。

1.2.1 面向连接编程模型

在这种编程模型下，当服务器程序的套接字创建并初始化结束时，它先进入休眠状态，直到有客户机向该服务器程序提出连接请求。这时，服务器程序被“唤醒”并开始响应客户机提出的连接请求，连接建立成功后双方相互发送并接收数据，在数据传输完毕时，双方再分别关闭连接并释放因创建套接字而占用的资源。面向连接编程模型如图 1-1 所示。

1.2.2 无连接编程模型

使用无连接编程模型在传输数据之前，无须事先建立连接，有数据就进行发送，但却不对数据的顺序和正确性负责，相对于面向连接模型，它的传输效率较高，因为它缺少了对数据的确认和确保数据有效的冗余字段。它的编程模型如图 1-2 所示。

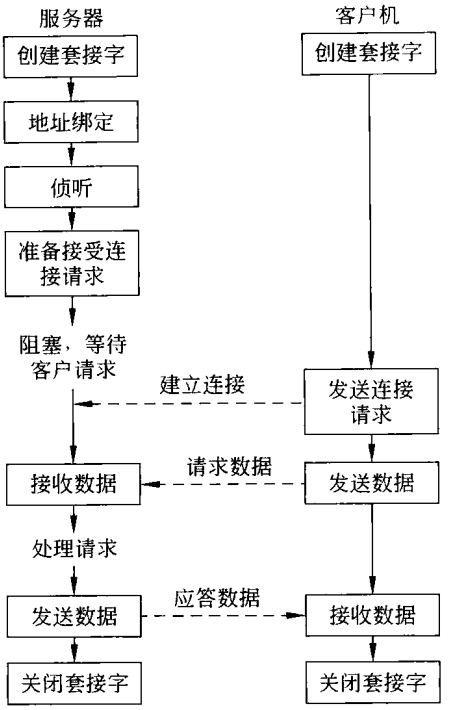

图 1-1　面向连接编程模型

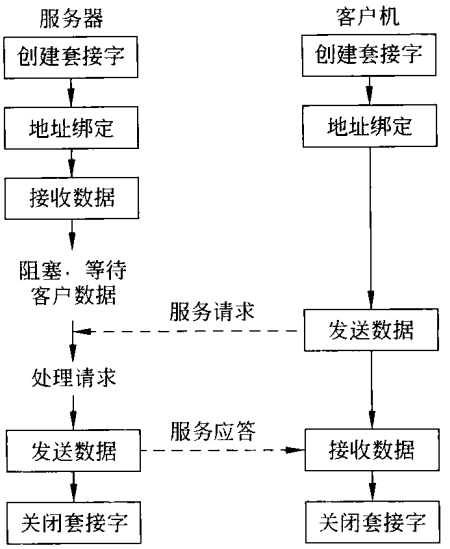

图 1-2　无连接编程模型

1.2.3　几个基本概念

1. 带外数据

带外数据，即 TCP 紧急数据，是指相连的每一对流式套接口间的一个逻辑上独立的传输通道。带外数据是独立于普通数据传送给用户的，这就要求带外数据设备必须支持每一时刻至少有一个带外数据消息被可靠地传送。这一消息至少包含一个字节，并且在任何时刻仅有一个带外数据信息等候发送。对于仅支持带内数据的通信协议来说（例如紧急数据是与普通数据在同一序列中发送的），系统通常把紧急数据从普通数据中分离出来单独存放，这就允许用户在顺序接收紧急数据和非顺序接收紧急数据之间做出选择（非顺序接收时可以省去缓存重叠数据的麻烦）。

在特殊情况下，某一个应用程序也可能喜欢处理紧急数据，即把其作为普通数据流的一部分，这可以通过设置套接口选项中的 SO_OOBINLINE 来实现。在这种情况下，应用程序可能希望确定未读数据中哪些是“紧急”的（“紧急”这一术语通常应用于线内带外数据）。为了达到这个目的，在 Winsock 的实现中就要在数据流中保留一个逻辑记号来指出带外数据从哪一点开始发送，一个应用程序可以使用 ioctrlsocket()函数访问 SIOCATMARK 标志来确定在记号之前是否还有未读入的数据。应用程序可以使用这一记号与对方重新进行同步。

2. 广播

数据报套接口可以用来向许多系统支持的网络发送广播数据包。要实现这种功能，网络本身必须支持广播功能，因为系统软件本身并不提供对广播功能的任何模拟。广播信息

将会给网络造成极重的负担，因为它们要求网络上的每台主机都为它们服务，所以发送广播数据包的能力被限制在那些显式标记了允许广播的套接口中。广播通常是为了如下两个原因而使用的。

(1) 一个应用程序希望在本地网络中找到一个资源，而应用程序对该资源的地址先前并不知道。

(2) 一些重要的功能，例如路由要求把它们的信息发送给所有可以找到的邻机。

被广播信息的目的地址取决于这一信息将在何种网络上广播。Internet 域支持一个速记地址(INADDR_BROADCAST)用于广播。内于使用广播以前必须捆绑一个数据报套接口，所以所有收到的广播消息都带有发送者的地址和端口。

某些类型的网络支持多种广播的概念，例如 IEEE 802.5 令牌环结构便支持链接层广播指示，它用来控制广播数据是否通过桥接器发送。Winsock 规范没有提供任何机制用来判断某个应用程序是基于何种网络之上的，而且也没有任何办法来控制广播的语义。

3. 字节顺序

不同的计算机可能会用不同的字节顺序存储数据。如 Intel 处理器的字节顺序是和 DEC VAX 处理器的字节顺序一致的，但是它与 6800 型处理器以及 Internet 的顺序却是不同的，所以用户在使用时要特别小心以保证正确的顺序。

任何 Winsock 函数对 IP 地址和端口号的引用以及传送给 Winsock 函数的 IP 地址和端口号均是按照网络顺序组织的，这也包括了 sockaddr_in 这一数据结构中的 IP 地址域和端口域。

如果用户输入一个数，而且指定使用这一端口号，则应用程序必须在使用它建立地址以前把它从主机顺序转换成网络顺序，这种转换可以通过使用 htons()函数来完成。相应地，如果应用程序希望显示包含于某一地址中的端口号(例如从 getpeername()函数中返回的)，这一端口号就必须在被显示前从网络顺序转换到主机顺序，该操作可以通过使用 ntohs()函数来完成。同样，关于 IP 地址可以类似处理。

由于 Intel 处理器和 Internet 的字节顺序是不同的，因而上述的转换是无法避免的，应用程序的编写者应该使用作为 Winsock API 一部分的标准的转换函数，而不要使用自己的转换函数代码。因为将来的 Winsock 实现有可能在主机字节顺序与网络字节顺序相同的机器上运行，因此只有使用标准的转换函数的应用程序才是可移植的。

4. 阻塞和非阻塞

在 Socket 网络编程中，套接字可根据需要被设为阻塞模式和非阻塞模式。当处于阻塞模式时，Socket 会一直等待下去，直到操作完成。例如，当调用 Receive 功能函数时，Socket 会被阻塞直到有新的数据到达；而当 Socket 处于非阻塞模式时，调用会立即返回，通常这些调用都会返回“失败”。Winsock 的 I/O 模型可以帮助应用程序判断一个套接字何时可供读写。

套接字通常都会和线程搭配使用。如果采用阻塞套接字则通常会和多线程相搭配，在不同的线程中使用不同的套接字，这样即使某个线程中的套接字被阻塞，也不会影响其他线程套接字的使用；而当采用非阻塞模式时，则没有这方面的限制。

1.3 基本操作函数

1.3.1 Winsock 的启动和终止

1. 启动——WSAStartup()

由于 Winsock 在被调用时是以动态链接库 DLL 形式实现的，所以在它初始化时应首先调用 WSAStartup()函数，对 Winsock DLL 进行创始化，确定被调用的 Winsock 的版本号，并为此分配必要的资源。现在来看一下以下程序代码：

```
//-------------------------------------------------------------
    WORD wVersionRequested;                    //应用程序所请求的 Winsock 版本号
    WSADATA wsaData;                           //用来返回 Winsock 实现的细节信息
    Int err;                                   //出错代码
    //生成版本号 1.1
    wVersionRequested=MAKEWORD(1,1);
    //调用初始化函数
    err=WSAStartup(wVersionRequested,&wsaData);
    //通知用户找不到合适的 DLL 文件
    if(err!=0){return;}
    //确认返回的版本号是不是客户请求的 1.1
    if(LOBYTE(wsaData.wVersion)!=1||HYBYTE(wsaData.wVersion)!=1){
        WSACleanup();
        return;
    }
    /* 至此,可以确认初始化成功,Winsock DLL 可用 */
//-------------------------------------------------------------
```

WSAStartup()函数的原型如下：

```
int WSAStartup(WORD wVersionRequested, LPWSADATA lpWSAData);
```

参数说明(以后将用[IN]表示输入参数，[OUT]表示输出参数)：

(1) 参数 wVersionRequested [IN]：用于存储要加载的 Winsock 库的版本，一般高位字节用于存储 Winsock 库的副版本，而低位字节则用来存储主版本。而前面所用到的宏 MAKEWORD(X, Y)，则用来构造一个完整的版本信息。

(2) 参数 lpWSAData [OUT]：是一个指向 LPWSADATA 结构的指针，该结构包含了加载库版本的相关信息，用来返回 Winsock API 实现的细节信息。

数据类型如下：

```
Typedef struct WSAData{
    WORD            wVersion;
    WORD            wHignVersion;
    Char            szDescrition[WSADESCRIPTION_LEN+1];
    Char            szSystemStatus[WSASYS_STATUS_LEN+1];
    Unsigned short  iMaxSockets;
```

```
    Unsigned short   iMaxUdpDg;
    Char FAR *       lpVendorInfo;
}WSADATA, * LPWSADATA;
```

其中，wVersion 字段存储当前使用的版本信息；而 wHignVersion 则为 Winsock 库的最高版本；szDescrition 和 szSystemStatus 被闲置，一般不用；iMaxSockets 为可同时打开的套接字数目；iMaxUdpDg 则为数据报的最大长度；lpVendorInfo 是为指定厂商信息预留的。

返回值：如果调用成功，则返回 0；否则，返回错误信息。

WSAStartup()函数可能返回的错误代码主要有以下 6 个。

(1) WSASYSNOTREADY：网络通信依赖的网络子系统没有准备好。

(2) WSAVERNOTSUPPORTED：找不到所需的 Winsock API 相应的动态连接库。

(3) WSAEINVAL：DLL 不支持应用程序所需的 Winsock 版本。

(4) WSAEINPROGRESS：正在执行一个阻塞的 WinSock 1.1 操作。

(5) WSAEPROCLIM：已经达到 Winsock 支持的任务数上限。

(6) WSAEFAULT：参数 lpWSAData 不是合法指针。

2. 终止——WSACleanup()

当网络通信完成，套接字被关闭后，需要调用 WSACleanup()函数终止对 Winsock DLL 的使用，并释放资源。任何打开的并已建立连接的 SOCK_STREAM 类型套接口在调用 WSACleanup()时会重置，而已经由 closesocket()关闭，但仍有要发送的悬而未决数据的套接口则不会受影响，该数据仍然继续发送。它的函数原型如下：

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

该函数没有任何参数，当执行成功后，返回 0，否则返回 SOCKET_ERROR。

WSACleanup()函数可能返回的错误代码主要有以下 3 种。

(1) WSANOTINITIALISED：使用本函数前必须要进行一次成功的 WSAStartup()调用。

(2) WSAENETDOWN：Winsock 实现已经检测到网络子系统故障。

(3) WSAEINPROGRESS：一个阻塞的 Winsock 操作正在进行。

对应于一个任务进行的每一次 WSAStartup()函数调用，必须有一个 WSACleanup()函数调用，就像括号一样，成对出现。但只有最后的 WSACleanup()做实际的清除工作；前面的调用仅仅将 Winsock DLL 中的内置引用计数减 1。在一个多线程的环境下，WSACleanup()中止 Winsock 在所有线程上的操作。一个简单的应用程序为确保 WSACleanup()调用了足够的次数，可以在一个循环中不断调用 WSACleanup()，直至返回 WSANOTINITIALISED 为止。

1.3.2 Winsock 的创建、绑定与关闭

1. 创建——socket()

套接字的创建非常简单，只需调用 socket()函数即可，过程如下：

```
//-------------------------------------------------------------
SOCKET sock;
```

```
sock=socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
//如果创建失败,返回错误信息,关闭套接字
if(sock==INVALID_SOCKET)
{
    closesocket(sock);
    return-1;
}
//-------------------------------------------------------------
```

socket()函数的原型如下：

```
SOCKET socket(int af, int type, int protocol);
```

参数说明：

(1) 参数 af [IN]：指定所创建的套接字的通信域，即指定应用程序使用的通信协议的协议簇。因为 Winsock1.1 只支持在 Internet 域通信，此参数只能取值为 AF_INET，这也就指定了此套接字必须使用 Internet 的地址格式。

(2) 参数 type [IN]：用于指定套接字的类型，若取 SOCK_STREAM 则表示要创建的套接字是流式套接字，若取 SOCK_DGRAM 则创建的是数据报套接字。在 Internet 域，这个参数实际指定了套接字使用的传输层协议。

(3) 参数 protocol [IN]：用来指定套接字使用的协议，一般采用默认值 0，表示让系统根据地址格式和套接字类型，自动选择一个合适的协议，如 TCP/IP 协议。

返回值：如果调用成功，就创建了一个新的套接字，并返回它的描述符。在以后对该套接字的操作中，都要借助这个描述符；否则返回 SOCKET_ERROR，表示创建套接字出错。应用程序可以调用 WSAGetLastError()函数获取相应的错误代码。

socket()函数可能返回的错误代码主要有以下 6 种。

(1) WSAEAFNOSUPPORT：不支持所指定的通信域或地址簇。

(2) WSAEMFILE：没有可用的套接字描述符，说明创建的套接字数目已超过限额。

(3) WSAENOBUFS：没有可用的缓冲区，无法创建套接字。

(4) WSAEPROTONOSUPPORT：不支持指定的协议。

(5) WSAEPROTOTYPE：指定的协议类型不适用于本套接口。

(6) WSAESOCKTNOSUPPORT：本地址簇中不支持该类型的套接字。

2. 绑定——bind()函数

绑定是将本地地址附加到所创建的套接字上以便能够有效地标识套接字的过程。

bind()函数的原型如下：

```
int bind(SOCCKET s, const struct sockaddr * name, int namelen);
```

参数说明：

(1) 参数 s[IN]：未经绑定的套接字描述符，是由 socket()函数返回的，要将它绑定到指定的网络地址上。

(2) 参数 name[IN]：一个指向 sockaddr 结构变量的指针，所指结构中保存着特定的网络地址，就是要把套接字 s 绑定到这个地址上。