ELSEVIER

(第2版)

TCP/IP Sockets in C Second Edition

TCP/IP Sockets 编程
(C语言实现)

Michael J. Donahoo Kenneth L. Calvert 著
陈宗斌 等 译

清华大学出版社

ELSEVIER

TCP/IP Sockets in C Second Edition

内容简介

本书为开发成熟且功能强大的Web应用程序提供所需的知识和技巧。本书以教学指南的方式，帮助读者掌握在C语言环境下，用套接字实现客户-服务器项目开发的任务和技术。本书的本版次增加了对最新技术的介绍，如对IPv6的支持，以及更详细的编程策略等内容。

本书内容简明扼要、示例丰富，既可作为高等学校网络编程课程的教学参考书，也是网络开发人员进行应用程序开发的常备参考书。

作者介绍

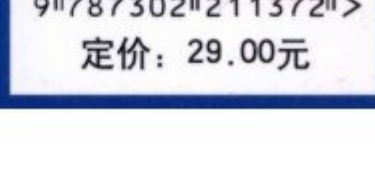
Michael J. Donahoo

Baylor大学副教授，为本科生和研究生讲授网络技术课程。他编写了多本关于各种语言的套接字编程图书，以及一本SQL方面的图书。

Kenneth L. Calvert

Kentucky大学教授，在该大学里，他主要讲授计算机网络系统课程，具有20多年的TCP/IP套接字编程经验。

ISBN 978-7-302-21137-2



(第2版)

TCP/IP Sockets in C Second Edition

TCP/IP Sockets 编程
（C语言实现）

Michael J. Donahoo Kenneth L. Calvert 著

陈宗斌 等 译

清华大学出版社
北京

声明

我们可以提供整本电子书的制作服务，价格便宜，如有需要请与我们联系。

本店淘宝网址：**http://shop61770214.taobao.com/** 店名：淘书**market**

永久有效QQ：909073796 （任何问题都可以通过909073796@qq.com进行邮件联系！）

本书由淘书整理制作，本店提供电子书定制服务目的在于给读者提供一个预览书籍的机会，请各位朋友在下载后小时之内自行删除，如确实需要此书，请购买正版图书。

本店核心业务-图书定制

图书定制链接：**http://item.taobao.com/item.htm?id=10280152022**

图书定制介绍：所有在国内发行过的书一般来说，只要半年前出版的基本都可以订制。书籍格式为清晰的，不加密，无任何限制的扫描版**pdf**文件，适合于电脑、电纸书、**ipad**、**iphone**、黑莓等所有支持**pdf**格式的阅读设备。一般**24**小时内发货！

图书定制方法： 通过旺旺联系或**QQ**以及邮件 ，将您需要的书籍“推荐去当当，卓越等网络搜索”网页发给我们，我们会在小时之内回复您是否可以定制。

图书定制价格： 价格按书籍正文页数来计算，**300**页以下是**4**元，**300**页以上每**100**页加**1**元，不足**100**页按**100**页计算。

注：电子书制作费时费力，而且都是有成本的，为了维持下去，故不能免费提供给大家整本书籍来阅读，还请见谅。

TCP/IP Sockets in C, Second Edition
Michael J. Donahoo, Kenneth L. Calvert
ISBN: 9780123745408

Authorized Simplified Chinese translation edition published by the Proprietor.
ISBN: 9789812724137

北京市版权局著作合同登记号 图字: 01-2009-4884 号

图书在版编目(CIP)数据

TCP/IP Sockets 编程:C 语言实现: 第 2 版/(美)多纳霍(Donahoo M. J.),(美)卡尔弗特(Calvert K. L.)著;陈宗斌等译. —北京:清华大学出版社,2009.11
书名原文:TCP/IP Sockets in C,Second Edition
ISBN 978-7-302-21137-2

Ⅰ. T… Ⅱ. ①多… ②卡… ③陈… Ⅲ. C 语言—程序设计 Ⅳ. TP312

中国版本图书馆 CIP 数据核字(2009)第 175766 号

责任编辑: 龙启铭
责任校对: 徐俊伟
责任印制: 杨 艳

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: 010-62776969,c-service@tup.tsinghua.edu.cn
质 量 反 馈: 010-62772015,zhiliang@tup.tsinghua.edu.cn
印 刷 者: 清华大学印刷厂
装 订 者: 三河市李旗庄少明装订厂
经 销: 全国新华书店
开 本: 185×230 **印 张**: 12.25 **字 数**: 294 千字
版 次: 2009 年 11 月第 1 版 **印 次**: 2009 年 11 月第 1 次印刷
印 数: 1～3000
定 价: 29.00 元

本书如存在文字不清、漏印、缺页、倒页、脱页等印装质量问题,请与清华大学出版社出版部联系调换。
联系电话: 010-62770177 转 3103 产品编号: 033748-01

译者序

目前，Internet 已经广泛渗透到人们的日常生活中，并且变得越来越重要。尽管现在有其他语言提供了对 Internet 的访问，人们对原始的基于 C 的 Berkeley Sockets 仍然有很浓厚的兴趣。对于那些想要设计和构建使用 Internet（即使用 TCP/IP）的分布式应用程序的人来说，Sockets API 仍然很重要。并且此接口已经证明非常健壮，足以支持网际协议的新版本（IPv6），现在几乎所有的公共计算平台都支持 IPv6。

在编写本书的第 2 版时，与第 1 版相比，考虑到了两个重要的方面。第一，有一些主题需要更深入地介绍，另外一些主题则需要扩展。第二个考虑是人们日益接受并使用 IPv6，它现在实质上受到所有当前的最终系统平台的支持。本书这一版在这两个方面都做了有针对性的介绍。

本书内容简明扼要、示例丰富，非常适合于学习涉及编程的计算机网络初级课程的学生（研究生和大学生）以及希望编写他们自己的程序以便通过 Internet 通信的程序员阅读。本书要求读者具有 C 和 UNIX 方面的基本编程技能和经验，熟悉一些 C 语言的概念，比如指针和类型强制转换，并且应该基本了解数据的二进制表示。

参加本书翻译的人员有：陈宗斌、王馨、陈红霞、张景友、易小丽、陈婷、管学岗、王新彦、金惠敏、张海峰、徐晔、戴锋、张德福等。

由于时间紧迫，加之译者水平有限，错误在所难免，恳请广大读者批评指正。

第 2 版的前言

在我们编写本书的第 1 版时，在关于联网的大学课程中包括编程方面的内容不是很常见。现在看起来这似乎令人难以置信，目前，Internet 变得对我们的世界如此重要，并且讲授实用编程和真实协议示例的好处被如此广泛地接受。尽管现在有其他语言提供了对 Internet 的访问，人们对原始的基于 C 的 Berkeley Sockets 仍然有很浓厚的兴趣。用于联网的 Sockets API（应用程序编程接口）是 20 世纪 80 年代在加州大学伯克利分校为 UNIX 的 BSD 版本开发的——这种 UNIX 版本是现在称为开源项目的最初几个示例之一。

Sockets API 和 Internet 在许多竞争性协议族——IPX、Appletalk、DECNet、OSI、SNA 以及 TCP/IP（Transmission Control Protocol/Internet Protocal，传输控制协议/网际协议）——的世界中逐渐成长起来，并且 Sockets 被设计成支持所有这些协议。在我们编写本书第 1 版时，只有很少的几个协议族被广泛应用，今天这个数量甚至更少。然而，就像我们在本书第 1 版中所预计的，对于那些想要设计和构建使用 Internet（即使用 TCP/IP）的分布式应用程序的人来说，Sockets API 仍然很重要。并且此接口已经证明非常健壮，足以支持网际协议的新版本（IPv6），现在几乎所有的公共计算平台都支持 IPv6。

两个主要的考虑促使我们编写了本书的第 2 版。第一，依据我们自己的经验和其他人的反馈意见，我们发现一些主题需要更深入地介绍，另外一些主题则需要扩展。第二个考虑是人们日益接受并使用 IPv6，它现在实质上受到所有当前的最终系统平台的支持。在编写本书时，不可能使用 IPv6 与 Internet 上的大部分主机交换信息，但是有可能给其中的许多主机分配一个 IPv6 地址。尽管称 IPv6 将接管整个世界仍言之过早，但是开始编写应用程序以为之做好准备却正当其时。

相比第 1 版的变化

更新并相当大地扩展了大部分材料，添加了两章内容。相比第 1 版的主要变化如下。

- 涵盖了 IPv6。现在包括了三种代码：特定于 IPv4 的代码、特定于 IPv6 的代码和通

用代码。后面几章中的代码设计成用于双栈机器上的任何一种协议版本。

- 新增了关于用 C++进行套接字编程的额外一章内容（由 David B. Sturgill 撰稿）。PracticalSocket 库提供了基本套接字功能的包装器。这些允许教师给没有 C 编程背景的学生讲授套接字编程，方法是给他们提供一个库，然后逐一剖析各层知识。学生在理解了地址/端口和客户/服务器之后，可以立即开始进行开发。以后可以通过剖析包装器代码内部的工作原理，向他们展示套接字编程的细节。那些讲授涉及联网的主题（例如，操作系统）的教师可以使用库，并且只是有选择地剖析其内部的工作原理。
- 更深入地介绍了用于组织发送和接收消息的代码的数据表示问题和策略。根据我们的教学经验，发现学生极少理解在内存中如何实际地存储数据[1]，因此我们尝试对这个问题进行更多的讨论以作为补偿。与此同时，国际化的重要性也与日俱增，因此本书包括了对广泛字符和编码的基本介绍。
- 省略了参考一节的内容。组成 Sockets API 的大多数函数的描述被集合进前面几章中。不过，由于有如此之多的在线参考信息资源（包括“参考手册（man page）”）可用，我们选择省略了 API 的完整列表，以便于进行更多的代码演示。
- 突出了重要而细微的事实和警告。排版设备突出了重要的概念和信息，在第 1 版中可能遗漏了这些信息。

尽管本书涵盖的范围有所扩展，但是我们没有把所有的一切都纳入本书中（甚至那些要求包括的内容也是如此）；有些主题需要参阅更全面的教材（或者将在下一版中介绍），这样一些主题的例子有原始套接字以及利用 WinSock 编程。

本书读者对象

我们最初编写本书的目的是，当我们希望学生学习套接字编程时，可以给他们分发一些资料，使得我们不必占用宝贵的课堂时间来讲授它。自从本书第 1 版起的几年间，我们了解到对于一些主题学生需要许多帮助，而有些主题则不需要太多手把手的指导。我们还发现我们的图书至少会被那些希望了解主题的简要介绍的从业者赏识。因此，本书同时针对两类普通读者：学习涉及编程的计算机网络初级课程的学生（研究生和大学生），以及希望编写他们自己的程序以便通过 Internet 通信的从业者。对于学生，本书打算作为一种补充材料，而不是关于网络的主要教材。尽管本书第 2 版的篇幅比第 1 版长很多并且涵盖的范围也大得多，我们还是希望本书仍然被人们认为作为补充材料具有良好的价值。对于只想编写一些有用代码的从业者，应该把本书作为独立的导论——但是应该告诫这类读者的是，

1 我们推测这是由于在大学课程里 C++和 Java 的广泛应用，它们对程序员隐藏了这类细节。

本书将不会使他们成为专家。我们通过实践学习知识的哲学没有改变，我们采用的方法也没有改变，即我们只提供足以帮助读者开始自学的简明教程，而把全面的细节留给其他作者去完成。对于这两类读者对象，我们的目标是教给读者足够多的知识，以便他们可以开始试验并自学相关知识。

假定的背景

我们假定读者具有 C 和 UNIX 方面的基本编程技能和经验。期望你熟悉一些 C 语言的概念，比如指针和类型强制转换，并且应该基本了解数据的二进制表示。一些示例将被分成应该单独编译的文件；我们假定你可以处理这项任务。

下面是一个小测试：如果你可以设法想出下面的代码段用于做什么，那么在理解本书中的代码方面应该没有什么问题：

```
typedef struct {
   int a;
   short s[2];
} MSG;

MSG *mp, m = {4, 1, 0};
char *fp, *tp;
mp = (MSG *) malloc(sizeof(MSG));
for (fp = (char *)m.s, tp = (char *)mp->s; tp < (char *)(mp+1);)
    *tp++ = *fp++;
```

如果不理解这个代码段，不要绝望（我们的代码中没有如此费解的内容），但是你可能想查阅自己最喜爱的 C 编程书籍，以便搞清楚这里将发生什么事情。

你还应该熟悉进程/地址空间、命令行参数、程序终止以及常规文件输入和输出等 UNIX 概念。第 4 章和第 6 章中的材料假定读者掌握了 UNIX 的更高级一些的知识。事先掌握一些网络方面的概念（比如协议、地址、客户和服务器）将是有帮助的。

平台需求与可移植性

我们的介绍是基于 UNIX 的。在开发本书时，好几个人强烈要求同时包括进针对 Windows 以及 UNIX 的代码。由于多方面的原因我们不可能这样做，包括为本书设定的目标篇幅长度（和定价）。

对于那些只能访问 Windows 平台的人，请注意本书前面几章中的示例需要进行最低限度的修改以便使用 WinSock（必须在程序开头更改包括文件并添加一个设置调用，并在程

序末尾添加一个清理调用）。其他大多数示例还需要额外一些非常少的修改。不过，有些示例是如此依赖于 UNIX 编程模型，以至于把它们移植到 WinSock 是没有意义的。可以从本书的 Web 站点（www.elsevierdirect.com/companions/9780123745408）上获取其他示例的为 WinSock 准备的版本，以及所需的代码修改的详细说明。另请注意：通过在用于 Windows 的 Cygwin UNIX 库程序包（可以在线获取它）下执行最少量的修改，就可以使几乎所有的示例代码工作。

对于本书第 2 版，我们采纳了 C99 语言标准。这个语言版本受到大多数编译器的支持，并且提供了如此之多的可以改进可读性的优点——包括行定界的注释、固定大小的整数类型，以及出现在块中任意位置的声明——以至于我们不能调整它，而只是使用它。

我们的代码利用了“Basic Socket Interface Extensions for IPv6”[6]。这些扩展当中有一个新的、不同的接口连接到名称系统。由于我们完全依赖于这个新的接口（getaddrinfo()），我们的通用代码可能不会在一些较老的平台上运行。不过，我们期望大多数现代系统将非常好地运行我们的代码。

本书中包括的示例程序全都在*NIX 和 MacOS 中进行了测试（应该可以不加修改地编译和运行）。头（.h）文件的位置和依赖性不是非常标准，可能需要在你的系统上进行一些修改。套接字选项支持也因系统的不同而有广泛的不同；我们尽量关注那些受到最普遍支持的套接字选项。参考 API 文档，了解系统的特定细节（API 文档的含义是指你的系统的“参考手册”。要了解这方面的情况，可以输入“man man”，或者使用你最喜爱的 Web 搜索工具）。

要知道的是，尽管我们努力实现基本级别的健壮性，但是我们的代码示例的主要目标是用于教学，并且代码不具有生产质量。出于简洁性和清晰性考虑，我们牺牲了一些健壮性，尤其是在通用的服务器代码中（事实证明：编写可以在 IPv4 和 IPv6 协议配置的各种组合下工作的服务器同时最大化各种环境下成功的客户连接的可能性是很重要的）。

本书将不会使你成为一名专家

我们希望本书第 2 版将会被用作一种参考资源，甚至对那些已经相当了解套接字的读者也是如此。与第 1 版一样，我们在编写这一版时也学到了一些知识。但是要成为一名专家需要多年的经验，以及学习其他更全面的资源[3、16]。

本书第 1 章旨在给你提供刚好够用的总体知识，以便让你准备好编写代码。第 2 章说明了如何使用 IPv4 或 IPv6 编写 TCP 客户和服务器。第 3 章说明如何使客户和服务器使用网络的名称服务，还描述了如何使它们独立于 IP 版本。第 4 章介绍了 UDP（User Datagram Protocol，用户数据报协议）。第 5 章和第 6 章提供了编写更多程序所需的背景知识，而第 7 章则把 Sockets 实现中的一些内容与 API 调用关联起来；这 3 章内容实质上是独立的，可

以任意顺序介绍它们。最后，第 8 章展示了一个 C++类库，它提供了对套接字功能的简化访问。

在全书中，某些语句将会高亮显示，比如：**本书将不会使你成为一名专家！**我们的目标是：使你注意那些细微而重要的事实和思想，在第一次阅读时可能会遗漏它们。页边距中的标记告诉你“注意”以粗体显示的任何内容。

致谢

正是由于许多人的贡献，本书才得以问世。除了在编写本书第 1 版时帮助过我们的那些人（Michel Barbeau、Steve Bernier、Arian Durresi、Gary Harkin、Ted Herman、Lee Hollaar、David Hutchison、Shunge Li、Paul Linton、Ivan Marsic、Willis Marti、Kihong Park、Dan Schmitt、Michael Scott、Robert Strader、Ben Wah 和 Ellen Zegura）之外，我们还要特别感谢 David B. Sturgill 和 Bobby Krupczak，David B. Sturgill 撰写了第 8 章中的代码和正文内容，Bobby Krupczak 则帮助审阅了本书第 2 版的草稿。最后，要感谢 Morgan Kaufmann/Elsevier 的各位工作人员——我们的编辑 Rick Adams、助理编辑 Maria Alonso 和项目经理 Melinda Ritchie，感谢你们的耐心、帮助，以及对我们的图书质量的关心。

反馈

我们非常有兴趣清除错误，这样可以改进将来版本/印刷的质量，因此如果你发现任何错误，请给我们中的任何一个人发送电子邮件。我们将在本书的 Web 页面上维护一份勘误表。

我们的电子邮件地址如下：

M.J.D.：jeff_donahoo@baylor.edu

K.L.C.：calvert@netlab.uky.edu

目录

第1章

简介

今天，人们可以使用计算机打电话、看电视、给他们的朋友发送即时消息、与其他人玩游戏，以及购买可以想到的大多数物品——从歌曲到汽车。程序通过 Internet 通信的能力使得所有这一切都成为可能。很难说现在可以通过 Internet 访问多少台个人计算机，但是我们可以安全地说这个数量在快速增长；用不了多长时间就会达到数十亿。而且，每天都在开发新的应用程序。在日益增长的带宽和访问量的推动下，在可预见的将来 Internet 的影响将继续升级。

一个程序怎样通过网络与另一个程序通信？本书的目标是：在 C 编程语言的环境下，开始让你理解这个问题的答案。长时间以来，C 都是实现网络通信软件所选的语言。的确，称为套接字的 API（application programming interface，应用程序编程接口）最初就是用 C 开发的。

不过，在深入研究套接字的细节之前，值得从总体上简要探讨一下网络和协议，看看我们的代码将适合于应用在什么位置。我们在本书中的目标不是讲授网络和 TCP/IP 如何工作——有许多针对此目的的优秀教材可以阅读[1、3、10、15、17]，而是介绍一些基本的概念和术语。

1.1 网络、分组和协议

计算机网络由通过通信信道互连的机器组成。我们把这些机器称为主机（host）和路由器（router）。主机是运行应用程序（比如 Web 浏览器、IM 代理或文件共享程序）的计算机。主机上运行的应用程序是网络的真正“用户”。路由器也称为网关（gateway），这种机器的职责是把信息从一条通信信道中继或转发（forward）到另一条通信信道。它们可能会运行程序，但是通常不会运行应用程序。出于我们的目的，通信信道（communication channel）是把字节序列从一台主机传送到另一台主机的工具；它可能是有线（例如，以太网）、无线（例如，WiFi）或其他连接。

声明

本书由*淘书 market* 整理制作，本店提供电子书定制服务目的在于给读者提供一个预览书籍的机会，请各位朋友在下载后 *24 小时之内自行删除*，如需要此书，*请购买正版图书*。

本店核心业务-图书定制

图书定制链接：http://item.taobao.com/item.htm?id=10280152022

图书定制介绍：所有在国内发行过的书一般来说，只要半年前出版的都可以订制。书籍格式为*清晰的，不加密，无任何限制*的扫描版 pdf 文件，适合于电脑、电纸书、ipad、iphone、黑莓等所有支持 pdf 格式的阅读设备。一般 48 小时内发货！

图书定制方法： 通过旺旺联系或 QQ 以及邮件 ，将您需要的书籍“推荐去当当，卓越等网络搜索”网页发给我们，我们会在 *12 小时之内*回复您是否可以定制。

图书定制价格： 价格按书籍正文页数来计算，300 页以下是 4 元，300 页以上每 100 页加 1 元，不足 100 页按 100 页计算。

本店淘宝网址：http://shop61770214.taobao.com/店名：淘书 market

永久有效 QQ：909073796 （任何问题都可以通过 909073796@qq.com 进行邮件联系！）

路由器很重要，这只是由于它并不实际用于把每一台主机直接连接到所有其他的主机，而是代之以将少数几台主机连接到一个路由器，再把该路由器连接到其他路由器，如此下去，从而构成网络。这种安排允许利用数量相对较少的通信信道连接每台机器：大多数主机只需要一条通信信道。不过，通过网络交换信息的程序不会直接与路由器交互，它们基本上感觉不到路由器的存在。

信息（information）的含义是指由程序构造和解释的字节序列。在计算机网络的环境中，这些字节序列一般称为分组（packet）。分组包含网络用于执行其任务的控制信息，有时也包括用户数据。一个例子是用于标识分组目的地的信息。路由器使用这种控制信息来查明如何转发每个分组。

协议（protocol）是关于由通信程序交换的分组及其含义的协定。协议说明如何构造分组——例如，目的地信息位于分组中的什么位置以及分组有多大——以及如何解释信息。协议通常设计成使用给定的能力解决特定的问题。例如，HTTP（HyperText Transfer Protocol，超文本传输协议）解决的是在存储或生成超文本对象的服务器之间传输这些对象的问题，Web 浏览器则使它们对用户可见并且有用。即时消息传递协议解决的是允许两个或更多的用户交换简短文本消息的问题。

实现一个有用的网络需要解决大量不同的问题。为了保持事情易于管理并且是模块化的，设计了不同的协议来解决不同的问题集。TCP/IP 就是这样一个解决方案的集合，有时称之为协议族（protocol suite）。它碰巧就是 Internet 中使用的协议族，但是它也可以在独立的专用网络中使用。今后，当我们谈论网络（network）时，指的就是使用 TCP/IP 协议族的任何网络。TCP/IP 协议族中的主要协议是 IP（Internet Protocol，网际协议）、TCP（Transmission Control Protocol，传输控制协议）和 UDP（User Datagram Protocol，用户数据报协议）。

事实证明：分层组织协议是有用的；TCP/IP 以及几乎所有其他的协议族都是这样组织的。图 1.1 显示了主机和路由器中的协议、应用程序与 Sockets API 之间的关系，以及数据从一个应用程序（使用 TCP）到另一个应用程序的流动。标记为 TCP 和 IP 的方框表示这些协议的实现。这样的实现通常驻留在主机的操作系统中。应用程序通过 Sockets API（以

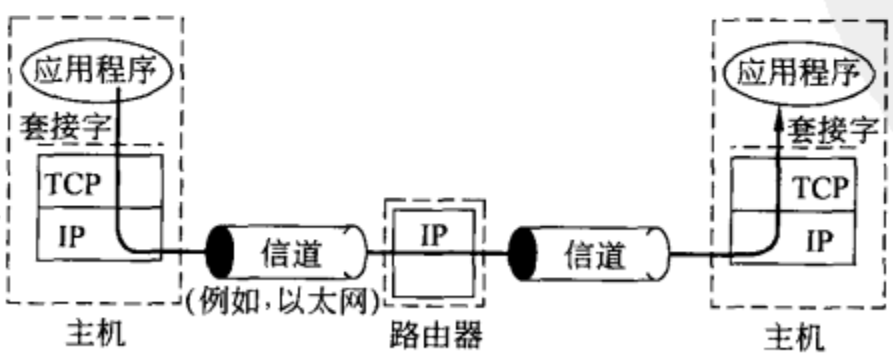


图 1.1　TCP/IP 网络

虚线表示）访问 UDP 和 TCP 提供的服务。箭头描绘了数据流动的方向：从应用程序开始，通过 TCP 和 IP 的实现，通过网络，并在另一端通过 IP 和 TCP 的实现进行备份。

在 TCP/IP 中，底下一层由底层通信信道（例如，以太网或拨号调制解调器连接）组成。这些信道由网络层（network layer）使用，这一层处理把分组转发到其目的地的问题（即路由器所做的工作）。TCP/IP 协议族中唯一的网络层协议是网际协议 IP；它解决的问题是使任意两台主机之间的信道和路由器序列看起来像是主机到主机之间的单独一条信道。

网际协议提供了数据报（datagram）服务：通过网络独立地处理并递送每个分组，就像通过邮政系统发送信件或包裹一样。为了使之工作，每个 IP 分组都必须包含其目的地的地址，就像把每个邮件包裹发送到某个人的地址一样（稍后将讨论关于地址的更多知识）。尽管大多数投递公司都会保证包裹的递送，但是 IP 是唯一一种“尽力而为”的协议：它尝试递送每个分组，但是在通过网络传输过程中它可能（并且偶尔会）丢失、重新排序或复制分组。

IP 上面一层称为传输层（transport layer）。它允许在两种协议中选择其一：TCP 和 UDP。每种协议都构建于 IP 提供的服务之上，但是它们采用不同的方式提供不同类型的传输，它们被具有不同需求的应用程序协议（application protocol）使用。TCP 和 UDP 具有一个共同的功能：寻址。回忆可知 IP 把分组递送到主机；显然，需要进行更细粒度的寻址以便把分组送到特定的应用程序，也许是同一台主机上使用网络的许多应用程序之一。TCP 和 UDP 都使用地址（称为端口号（port number））来确定主机内的应用程序。TCP 和 UDP 被称为端到端传输协议（end-to-end transport protocol），因为它们自始至终把数据从一个程序运送到另一个程序（而 IP 只把数据从一台主机运送到另一台主机）。

TCP 被设计成在 IP 提供的主机到主机信道中检测可能发生的丢失、复制分组及其他错误，并从中恢复过来。TCP 提供了可靠的字节流（reliable byte-stream）信道，因此应用程序不必处理这些问题。它是面向连接（connection-oriented）的协议：在把它用于通信之前，两个程序必须先建立一条 TCP 连接，这涉及在两台通信的计算机上的 TCP 实现之间完成握手消息（handshake message）的交换。使用 TCP 在许多方面也类似于文件输入/输出（I/O）。事实上，由一个程序写并由另一个程序读的文件就是通过 TCP 连接进行通信的合理模型。另一方面，UDP 不会尝试从 IP 经历的错误中恢复；它只是扩展 IP“尽力而为”的数据报服务，使得它在应用程序之间（而不是主机之间）工作。因此，使用 UDP 的应用程序必须准备好处理分组丢失、重新排序等问题。

1.2　关于地址

在邮寄信件时，在邮政业务可以理解的表格中提供收信人的地址。在可以通过电话与某个人交谈之前，必须给电话系统提供一个电话号码。与之类似，在一个程序可以与另一

个程序通信之前，它必须告诉网络某些信息以标识另一个程序。在 TCP/IP 中，它采用两份信息来标识一个特定的程序：IP 使用的 Internet 地址（Internet address）以及端口号（port number），后者是由传输协议（TCP 或 UDP）解释的额外地址。

Internet 地址是二进制数字。它们具有两种类型，对应于已经标准化的网际协议的两个版本。最常见的是版本 4（IPv4，[12]）；另一种是版本 6（IPv6，[5]），它刚刚开始部署。IPv4 地址的长度为 32 位，因为这正好足够用于标识 40 亿个不同的目的地，对于今天的 Internet 来说，它们确实不够大（这看起来似乎有很多地址，但是由于分配它们的方式，许多地址被浪费掉了。在全部 IPv4 地址空间中，目前已经分配了超过一半的地址）。因此，引入了 IPv6。IPv6 地址的长度为 128 位。

1.2.1 记下 IP 地址

在表示 Internet 地址以便于人类使用（与在程序内使用它们相对）方面，为 IP 的两个版本采用了不同的约定。按照惯例，将 IPv4 地址写为一组 4 个用句点隔开的十进制数字（例如，10.1.2.3），这称为点分四组（dotted-quad）表示法。点分四组字符串中的 4 个数字表示 Internet 地址的 4 个字节的内容——因此，每个部分都是一个 0～255 之间的数字。

另一方面，根据约定，16 字节的 IPv6 被表示为用冒号隔开的十六进制数字的组合（例如，2000:fdb8:0000:0000:0001:00ab:853c:39a1）。每一组数字表示 2 字节的地址；可以省略前导 0，因此上述示例中的第 5 组和第 6 组可以只呈现为:1:ab:。此外，只包含 0 的一个组序列可以完全省略（而在地址的其余部分中留下用于分隔它们的冒号）。因此，上述示例可以写为 2000:fdb8::1:00ab:853c:39a1。

从技术上讲，每个 Internet 地址都指的是主机与底层通信信道之间的连接——换句话说，就是网络接口。一台主机可能具有多个接口，例如，主机同时具有通往有线网络（以太网）和无线网络（WiFi）的连接是比较常见的。由于每条这样的网络连接都属于单个主机，Internet 地址就标识主机及其通往网络的连接。不过，反过来则不然，因为单个主机可能具有多个接口，并且每个接口可能具有多个地址（事实上，同一个接口可以同时具有 IPv4 和 IPv6 地址）。

1.2.2 处理两个版本

在编写本书第一版时，IPv6 尚未受到广泛的支持。今天，大多数系统都能够“开包即用”地支持 IPv6。为了顺利地从 IPv4 过渡到 IPv6，大多数系统都是双栈（dual-stack）系统，同时支持 IPv4 和 IPv6。在这样的系统中，每个网络接口（信道连接）可能至少具有一个 IPv4 地址和一个 IPv6 地址。

两个 IP 版本的存在使套接字程序员的生活变得复杂。一般来讲，在创建要通信的套接字时，将需要选择 IPv4 或 IPv6 作为底层协议。那么，怎样编写可以同时使用这两个版本

的应用程序呢？幸运的是，双栈系统通过支持这两个协议版本并且允许 IPv6 套接字与 IPv4 或 IPv6 应用程序通信，来处理互操作性。当然，IPv4 和 IPv6 地址区别很大；不过，可以使用 IPv4 映射的地址（IPv4 mapped address）将 IPv4 地址映射为 IPv6 地址。IPv4 映射的地址是通过在 IPv4 地址前添加 4 个字节的前缀::fff 而构成的。例如，132.3.23.7 的 IPv4 映射的地址是::ffff:132.3.23.7。为了便于人类阅读，通常用点分四组表示法书写最后 4 个字节。在第 3 章中将更详细地讨论协议互操作性。

不幸的是，具有 IPv6 Internet 地址并不足以让你能够通过 Internet 与所有其他支持 IPv6 的主机通信。为此，还必须与 ISP（Internet Service Provider，Internet 服务提供商）协商，要求提供 IPv6 转发服务。

1.2.3　端口号

我们以前提过，在把消息传送到程序时需要采用两段地址。TCP 或 UDP 中的端口号总是相对于 Internet 地址解释的。回到我们前面的类比上来，端口号对应于给定街道地址（比如大型建筑物）的房间号。邮政业务使用街道地址把信件投递到邮箱；然后，任何清空邮箱的人负责把信件送到建筑物内正确的房间。或者考虑一家具有内部电话系统的公司：要与该公司内的某个人通话，首先要拨打该公司的主机电话号码以连接到内部电话系统，然后拨打你希望与之通话的某个人的特定电话的分机号。在这些类比中，Internet 地址就是街道地址或公司的主机电话号码，而端口则对应于房间号或电话分机号。端口号在 IPv4 和 IPv6 中是相同的：即 16 位无符号的二进制数字。因此，每个端口号的范围在 1～65 535 之间（0 是预留的）。

1.2.4　特殊地址

在 IP 的每个版本中，都定义了某些特殊用途的地址。其中值得知道的地址是回送地址（loopback address），它总是被分配给特殊的回送接口（loopback interface），回送接口是简单地把传输的分组正确地回送给发送者的虚拟设备。回送接口对于测试非常有用，因为发送给该地址的分组将立即返回到目的地。而且，它存在于每一台主机上，甚至当计算机没有其他的接口（即未连接到网络）时也可以使用它。IPv4 的回送地址是 127.0.0.1；[1]，对于 IPv6，它是 0:0:0:0:0:0:0:1（或者只是::1）。

另一组 IPv4 地址是为特殊用途预留的，包括那些为“专用”预留的地址。这组地址包括以 10 或 192.168 开头的所有 IPv4 地址，以及那些第一个数字是 172 并且第二个数字在 16～31 之间的地址（对于 IPv6 没有对应的类别）。这些地址最初被指定在不属于全球 Internet 一部分的专用网络中使用。今天，它们通常用在通过 NAT（network address translation，网

1　从技术上讲，以 127 开头的任何 IPv4 地址都应该是回送地址。

络地址转换）[7]设备连接到 Internet 的家庭和小型办公室网络中。这种设备充当路由器，用于在转发分组中转换（重写）其中的地址和端口。更确切地讲，它把其接口之一上的分组中的（专用地址，端口）对映射到其他接口上的（公共地址，端口）对。这允许一小组主机（例如，家庭网络上的那些主机）有效地“共享”单个 IP 地址。这些地址的重要性在于：不能从全球 Internet 到达它们。如果在一台具有专用类别中的地址（例如，在家庭网络上）的机器上试验本书中的代码，并且尝试与不具有这些地址之一的另一台主机通信，一般不会成功，除非具有专用地址的主机发起通信——即使这样也可能会失败。

相关类别包含链路本地（link-local）或“自动配置”地址。对于 IPv4，这类地址以 169.254 开头。对于 IPv6，其前 16 位块是 FE80、FE90、FEA0 或 FEB0 的任何地址都是链路本地地址。这些地址只能用于连接到同一个网络的主机之间的通信；路由器将不会转发把这类地址作为其目的地的分组。

最后，另一个类别包含多播（multicast）地址。常规的 IP（有时称为“单播”）地址指单一目的地，而多播地址潜在地指任意数量的目的地。多播是一个高级主题，将在第 6 章中简要介绍它。在 IPv4 中，点分四组格式的多播地址的第一个数字在 224～239 之间。在 IPv6 中，多播地址以 FF 开头。

1.3 关于名称

你最有可能习惯于按名称（name）引用主机（例如，host.example.com）。不过，Internet 协议处理的是地址（二进制数字），而不是名称。你应该理解使用名称而不是地址是一种方便的特性，它独立于 TCP/IP 提供的基本服务——无需使用名称即可编写并使用 TCP/IP 应用程序。在使用名称标识通信端点时，系统将做一些额外的工作把名称解析为地址。由于两个原因，这个额外的步骤通常是值得做的。第一，与点分四组表示法（就 IPv6 而言，是十六进制的数字串）相比，人类显然更容易记住名称。第二，名称提供了某种级别的间接性，它把用户与 IP 地址变化隔离开。在编写本书第 1 版期间，Web 服务器 www.mkp.com 的地址改变了。由于我们总是按名称引用该 Web 服务器，www.mkp.com 将解析成当前 Internet 地址，而不是 208.164.121.48。IP 地址中的变化对于使用名称访问 Web 服务器的程序是透明的。

名称解析服务可以访问广泛来源的信息。两种主要的来源是 DNS（Domain Name System，域名系统）和本地配置数据库。DNS [8]是一种分布式数据库，它把像 www.mkp.com 这样的域名映射到 Internet 地址及其他信息；DNS 协议[9]允许连接到 Internet 的主机使用 TCP 或 UDP 从该数据库中检索信息。本地配置数据库一般是用于本地名称-Internet 地址映射的特定于操作系统的机制。

1.4　客户与服务器

在我们的邮政和电话系统的类比中，每次通信都是由一方发起的，发起方发送信件或者打电话，而另一方则通过发送回复信件或者拿起电话并交谈来响应发起者的联系。Internet 通信是类似的。客户（client）和服务器（server）就是指这些角色：客户程序发起通信，而服务器程序则被动地等待，然后响应联系它的客户。客户与服务器一起组成了应用程序（application）。客户与服务器描述了一种典型的情况，其中服务器使一种特定的能力（例如，数据库服务）可供能够与之通信的任何客户使用。

程序是充当客户还是充当服务器决定了它使用 Sockets API 与其对应方（peer）建立通信的一般形式（客户是服务器的对应方，反之亦然）。此外，客户-服务器的区别很重要，因为客户最初需要知道服务器的地址和端口，但是服务器则不然。利用 Sockets API，当服务器接收到来自客户的初始通信时，如果需要，它可以获悉客户的地址信息。这类似于打电话——为了被打电话，一个人不需要知道打电话的人的电话号码。与打电话一样，一旦建立了连接，服务器与客户之间的区别就消失了。

客户怎样查明服务器的 IP 地址和端口号呢？通常，客户知道它想要与之通信的服务器的名称——例如，通过像 http://www.mkp.com 这样的 URL（Universal Resource Locator，统一资源定位器），并使用名称解析服务获悉相应的 Internet 地址。

查找服务器的端口号则是一个不同的故事。从原理上讲，服务器可以使用任何端口，但是客户必须能够获悉它是什么。在 Internet 中，具有把众所周知的端口号分配给某些应用程序的惯例。IANA（Internet Assigned Number Authority，Internet 编号授权委员会）监督这种分配方式。例如，端口号 80 被分配给 HTTP（HyperText Transfer Protocol，超文本传输协议）。在运行 HTTP 客户浏览器时，它默认会尝试联系那个端口上的 Web 服务器。所有分配的端口号列表是由 Internet 编号授权委员会维护的（参见 http://www.iana.org/assignments/port-numbers）。

你可能听过客户-服务器的一种替代方案，即 P2P（peer-to-peer，点对点）方案。在 P2P 中，应用程序同时消费和提供服务，这与传统的客户-服务器体系结构不同，在这种体系结构中，服务器提供服务，客户则消费服务。事实上，P2P 节点有时也称为“servent”，它结合了 server 和 client 这两个单词。那么，你需要学习一组不同的技术给 P2P（而不是客户-服务器）编程吗？不是这样的。在 Sockets 中，客户与服务器之间的区别只在于谁建立初始连接，以及谁等待连接。P2P 应用程序通常会发起连接（到达现有的 P2P 节点）并接受连接（来自其他 P2P 节点）。在阅读本书后，你将能够像客户-服务器那样很好地编写 P2P 应用程序。

1.5 什么是套接字

套接字（socket）是一个抽象层，应用程序可以通过它发送和接收数据，其方式与打开文件句柄允许应用程序读、写数据到稳定的存储器非常相似。套接字允许应用程序插入到网络中，并与插入到同一个网络中的其他应用程序通信。由一台机器上的应用程序写到套接字中的信息可以被不同机器上的应用程序读取，反之亦然。

不同类型的套接字对应于不同的底层协议族以及协议族内的不同协议栈。本书只论及 TCP/IP 协议族。今天，TCP/IP 中的主要类型的套接字是流套接字（stream socket）和数据报套接字（datagram socket）。流套接字使用端到端协议（其下一层是 IP），从而提供可靠的字节流服务。TCP/IP 流套接字代表 TCP 连接的一端。数据报套接字使用 UDP（同样，其下一层是 IP），从而提供“尽力而为”的数据报服务，应用程序可以使用它来发送各个最大长度大约为 65 500 个字节的消息。流套接字和数据报套接字还受到其他协议族的支持，但是本书只论及 TCP 流套接字和 UDP 数据报套接字。TCP/IP 套接字是由 Internet 地址、端到端协议（TCP 或 UDP）和端口号唯一标识的。在学习本书的过程中，你将遇到将套接字绑定到地址的多种方式。

图 1.2 描绘了单个主机内的应用程序、套接字抽象层、协议和端口号之间的逻辑关系。关于这些关系要注意几点。第一，一个程序可以同时使用多个套接字。第二，多个程序可以同时使用同一个套接字抽象层，尽管这种情况不太常见。图 1.2 显示每个套接字都有一个关联的本地 TCP 或 UDP 端口，它用于把传入的分组指引到应该接收它们的应用程序。以前我们说过端口标识主机上的应用程序。实际上，端口标识主机上的套接字。不过，如图 1.2 所示，由于多个套接字可以与一个本地端口相关联，不能仅仅使用端口来标识套接字。对于 TCP 套接字，这种情况最常见；幸运的是，不需要理解这种细节即可编写使用 TCP 套接字的客户-服务器程序。第 7 章中将介绍关于这方面的完整知识。

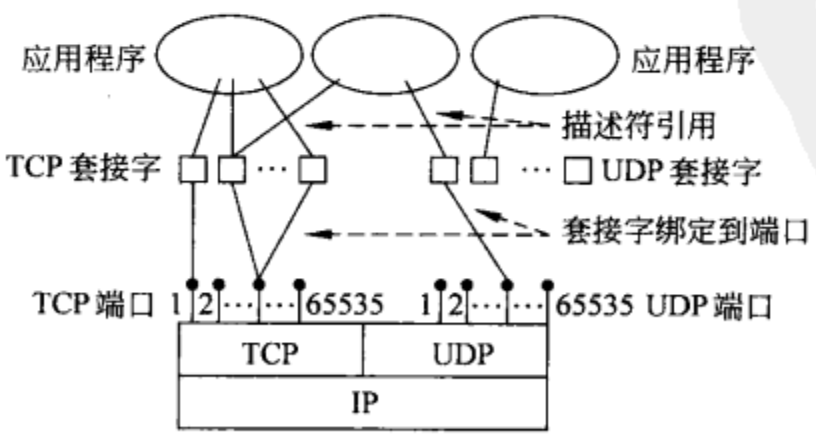


图 1.2 套接字、协议与端口

练习题

1. 在*NIX 中使用 ifconfig 命令或者在 Windows 中使用 ipconfig 命令报告你的 IP 地址。确定这些地址都是 IPv6 格式的地址。
2. 使用 hostname 命令报告你正在使用的计算机的名称。
3. 你可以查明任何直接连接的路由器的 IP 地址吗？
4. 使用 Internet 搜索试着了解关于 IPv5 所发生的事情。
5. 使用尽可能少的字符书写以下 IPv6 地址：2345:0000:0000:A432:0000:0000:0000:0023
6. 你能够想到不适合客户-服务器模型的真实的通信示例吗？
7. 你的家庭连接到了多少种不同类型的网络？有多少种网络支持双向传输？
8. IP 是一种“尽力而为”的协议，它要求把信息分解为数据报，它们可能会丢失、复制或重新排序。TCP 隐藏了所有这些细节，提供了一种可靠的服务，用于接收和递送完整的字节流。关于在 IP 之上提供 TCP 服务你可能会做什么？在 TCP 可用时，为什么任何人都会使用 UDP？

第 2 章

基本的 TCP 套接字

现在该学习编写你自己的套接字应用程序了。我们将从 TCP 开始。此时，你可能准备好动手编写一些实际的代码，因此我们首先将完成一个 TCP 客户和服务器的工作示例，然后将介绍基本 TCP 中使用的套接字 API 的详细信息。为了保持事情更简单，我们最初将介绍适用于一种特定的 IP 版本（即 IPv4）的代码，在编写本书时，它仍然是网际协议占主导地位的版本，这样就有了广泛的回旋余地。本章末尾介绍了编写客户和服务器的 IPv6 版本所需要的（少量）修改。第 3 章将演示独立于协议的应用程序的创建方法。

我们的示例客户和服务器实现了应答（echo）协议。它的工作方式如下：客户连接到服务器并发送它的数据；服务器简单地把它接收到的任何内容发送回客户并断开连接。在应用程序中，客户发送的数据是作为命令行参数提供的字符串。客户将打印它从服务器接收到的数据，因此可以看到返回的内容。许多系统出于调试和测试的目的都包括应答服务。

2.1 IPv4 TCP 客户

客户与服务器之间的区别很重要，因为在通信过程中的某些步骤中它们以不同的方式使用套接字接口。我们首先重点关注客户。它的职责是发起与被动等待联系的服务器之间的通信。

典型的 TCP 客户的通信涉及如下 4 个基本步骤。

（1）使用 socket()创建 TCP 套接字。

（2）使用 connect()建立到达服务器的连接。

（3）使用 send()和 recv()通信。

（4）使用 close()关闭连接。

TCPEchoClient4.c 是用于 IPv4 的 TCP 应答客户的实现。

TCPEchoClient4.c

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #include <unistd.h>
5 #include <sys/types.h>
6 #include <sys/socket.h>
7 #include <netinet/in.h>
8 #include <arpa/inet.h>
9 #include "Practical.h"
10
11 int main(int argc, char *argv[]) {
12
13 if (argc <3||argc> 4)           //Test for correct number of arguments
14   DieWithUserMessage("Parameter(s)",
15       "<Server Address> <Echo Word> [<Server Port>]");
16
17 char *servIP = argv[1];         //First arg; server IP address (dotted quad)
18 char *echoString = argv[2];     //Second arg: string to echo
19
20 //Third arg (optional): server port (numeric). 7 is well-known echo port
21 in_port_t servPort = (argc == 4) ? atoi(argv[3]) : 7;
22
23 //Create a reliable, stream socket using TCP
24 int sock = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
25 if (sock < 0)
26   DieWithSystemMessage("socket() failed");
27
28 //Construct the server address structure
29 struct sockaddr_in servAddr;            //Server address
30 memset(&servAddr, 0, sizeof(servAddr));//Zero out structure
31 servAddr.sin_family = AF_INET;          //IPv4 address family
32 //Convert address
33 int rtnVal = inet_pton(AF_INET, servIP, &servAddr.sin_addr.s_addr);
34 if (rtnVal == 0)
35   DieWithUserMessage("inet_pton() failed", "invalid address string");
36 else if (rtnVal < 0)
37   DieWithSystemMessage("inet_pton() failed");
38 servAddr.sin_port = htons(servPort);    //Server port
39
40 //Establish the connection to the echo server
41 if (connect(sock, (struct sockaddr *) &servAddr, sizeof(servAddr)) < 0)
42   DieWithSystemMessage("connect() failed");
```

```
43
44  size_t echoStringLen = strlen(echoString); //Determine input length
45
46  //Send the string to the server
47  ssize_t numBytes = send(sock, echoString, echoStringLen, 0);
48  if (numBytes < 0)
49    DieWithSystemMessage("send() failed");
50  else if (numBytes != echoStringLen)
51    DieWithUserMessage("send()", "sent unexpected number of bytes");
52
53  //Receive the same string back from the server
54  unsigned int totalBytesRcvd = 0;//Count of total bytes received
55  fputs("Received: ", stdout);    //Setup to print the echoed string
56  while (totalBytesRcvd < echoStringLen) {
57    char buffer[BUFSIZE];          //I/O buffer
58    /* Receive up to the buffer size (minus 1 to leave space for
59    a null terminator) bytes from the sender */
60    numBytes = recv(sock, buffer, BUFSIZE - 1, 0);
61    if (numBytes < 0)
62      DieWithSystemMessage("recv() failed");
63    else if (numBytes == 0)
64      DieWithUserMessage("recv()", "connection closed prematurely");
65    totalBytesRcvd += numBytes;    //Keep tally of total bytes
66    buffer[numBytes] = '\0';       //Terminate the string!
67    fputs(buffer, stdout);         //Print the echo buffer
68  }
69
70  fputc('\n', stdout);             //Print a final linefeed
71
72  close(sock);
73  exit(0);
74 }
```

TCPEchoClient4.c 做了以下事情。

（1）应用程序建立和参数解析：第 1～21 行。

- 包括文件：第 1～9 行。
  这些头文件声明了 API 的标准函数和常量。参考文档（例如，参考手册），了解用于系统上的套接字函数和数据结构的合适的包括文件。利用了自己的包括文件 Practical.h，它带有用于自己的函数的原型，将在下面描述它们。
- 典型的参数解析和可靠性检查：第 13～21 行。
  作为前两个参数传入要发回的 IPv4 地址和字符串。客户可以选择接受服务器端口

作为第三个参数。如果没有提供端口，客户将使用众所周知的应答协议端口 7。

（2）TCP 套接字的创建：第 23～26 行。

使用 socket()函数创建套接字。该套接字用于 IPv4（af_inet），使用称为 TCP（ipproto_tcp）的基于流的协议（sock_stream）。如果成功，socket()就会为套接字返回一个整数值的描述符或者“句柄”。如果套接字失败，它就会返回-1，并且会调用错误处理函数 DieWithSystemMessage()（以后将描述它），打印一个提供信息的提示并退出。

（3）准备地址并建立连接：第 28～42 行。

- 准备 sockaddr_in 结构用于存放服务器地址：第 29～30 行。
  为了建立套接字，必须指定要连接到的地址和端口。sockaddr_in 结构被定义为这种信息的“容器”。调用 memset()确保未显式设置的结构的任何部分都包含 0。
- 填充 sockaddr_in：第 31～38 行。
  必须设置地址族（AF_INET）、Internet 地址和端口号。函数 inet_pton()把服务器的 Internet 地址的字符串表示（以点分四组表示法作为命令行参数传入）转换为 32 位的二进制表示。以前把服务器的端口号从命令行字符串转换为二进制形式；调用 htons()（意指“host to network short”）确保根据 API 的需要对二进制值进行格式化（第 5 章中描述了这样做的原因）。
- 连接：第 40～42 行。
  connect()函数用于建立给定套接字与由 sockaddr_in 结构中的地址和端口标识的套接字之间的连接。由于 Sockets API 是通用的，需要把指向 sockaddr_in 地址结构（它特定于 IPv4 地址）的指针强制转换（cast）为泛型类型（sockaddr*），并且必须提供地址数据结构的实际大小。

（4）把应答字符串发送给服务器：第 44～51 行。

查明参数字符串的长度并保存它，以便以后使用。把指向应答字符串的指针传递给 send()调用；在应用程序启动时，把字符串本身存储在某个位置（像所有的命令行参数一样）。我们确实不关心它位于何处，而只需知道第一个字节的地址以及要发送多少字节（注意：我们不会发送位于参数字符串末尾（以及 C 语言中的所有字符串）的字符串末尾标记符（0））。如果成功，send()就会返回发送的字节数；否则，它就会返回-1。如果 send()失败或者发送了错误的字节数，我们必须处理错误。注意：这里将不会发生发送错误的字节数这种情况。然而，包括进针对这种情况的测试是一个好主意，因为在某些环境中可能发生这种错误。

（5）接收应答服务器的答复：第 53～70 行。

TCP 是一种字节流协议。这类协议的一种实现是不会保持 send()的边界。**通过在连接的一端调用 send()发送的字节可能不会通过在另一端单独调用一次 recv()而全都返回**（在第 7 章中将更详细地讨论这个问题）。因此需要反复接收字节，直至接收到的字节数与发送的

字节数相等为止。这个循环十有八九只会执行一次，因为来自服务器的数据事实上将一次性全部返回；不过，不保证这会发生，因此必须允许需要多次读取字节的可能性。编写使用套接字的应用程序的基本原则是：**对于另一端的网络和程序将要做什么事情，永远都不能做出假设。**

- 接收字节块：第 57～65 行。
  recv()会阻塞到数据可用为止，并且返回复制到缓冲区中的字节数，如果失败，则会返回–1。如果返回值为 0，则指示另一端的应用程序关闭了 TCP 连接。注意：传递给 recv()的大小参数将为添加 null 终止字符预留空间。
- 打印缓冲区：第 66～67 行。
  在接收到服务器发送的数据时将打印它。在接收到的每个数据块末尾添加 null 终止字符（0），以便 fputs()可以把它作为字符串处理。我们没有检查接收到的字节数是否与发送的字节数相等。服务器可能发送完全不同的内容（这取决于发送的字符串的长度），并将把它写到标准输出。
- 打印换行符：第 70 行。
  当接收到与发送的一样多的字节时，就会退出循环，并打印一个换行符。

（6）终止连接并退出：第 72～73 行。

close()函数通知远程套接字通信结束，然后释放分配给套接字的本地资源。

客户应用程序（实际上包括本书中的所有程序）利用了如下两个错误处理函数：

```
DieWithUserMessage(const char *msg, const char *detail)
DieWithSystemMessage(const char *msg)
```

这两个函数都把用户提供的消息字符串（msg）打印到 stderr，其后接着是详细的消息字符串；然后，它们利用一个错误返回代码调用 exit()，导致应用程序终止。唯一的区别是详细消息的来源。对于 DieWithUserMessage()，详细消息是用户提供的；对于 DieWithSystemMessage()，详细消息是由系统基于特殊变量 errno 的值提供的（该变量描述了系统调用的最近失败（如果有的话）的原因）。仅当错误情况是由设置 errno 的系统调用产生的时，才会调用 DieWithSystemMessage()（为了使程序保持简单，示例没有包含专门用于从错误中恢复的大量代码——它们只是简单地终止程序并退出。生产代码一般不应该如此轻易地放弃）。

偶尔，我们需要在不退出的情况下给用户提供信息；如果需要格式化能力，就使用 printf()；否则，就使用 fputs()。特别是，应该尽量避免使用 printf()输出固定的、预先格式化的字符串。**你永远也不应该做的一件事是：把从网络接收到的文本作为第一个参数传递给 printf()。它会引起严重的安全性问题，要代之以使用 fputs()。**

注意：DieWith...()函数是在头文件“Practical.h”中声明的。不过，这些函数的实际实

现包含在 DieWithMessage.c 文件中，应该编译该文件并把它与本书中的所有示例应用程序相连接。

DieWithMessage.c

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void DieWithUserMessage(const char *msg, const char *detail) {
5   fputs(msg, stderr);
6   fputs(": ", stderr);
7   fputs(detail, stderr);
8   fputc('\n', stderr);
9   exit(1);
10 }
11
12 void DieWithSystemMessage(const char *msg) {
13   perror(msg);
14   exit(1);
15 }
```

如果编译 TCPEchoClient4.c 和 DieWithMessage.c 创建程序 TCPEchoClient4，就可以与具有 Internet 地址 169.1.1.1 的应答服务器通信，如下所示：

```
% TCPEchoClient4 169.1.1.1 "Echo this!"
Received: Echo this!
```

为了让客户工作，就需要服务器。许多系统出于调试和测试的目的，包括了一个应答服务器；不过，出于安全性原因，这样的服务器最初通常是禁用的。如果不能访问应答服务器，也没有问题，因为我们将编写一个应答服务器。

2.2　IPv4 TCP 服务器

我们现在把注意力转向构造 TCP 服务器。服务器的职责是建立通信端点，并被动等待来自客户的连接。对于基本的 TCP 服务器通信，要执行如下 4 个常规的步骤。

（1）使用 socket()创建 TCP 套接字。

（2）利用 bind()给套接字分配端口号。

（3）使用 listen()告诉系统允许对该端口建立连接。

（4）反复执行以下操作：

- 调用 accept()为每个客户连接获取新的套接字。