TURING 图灵程序设计丛书 微软技术系列

Apress®

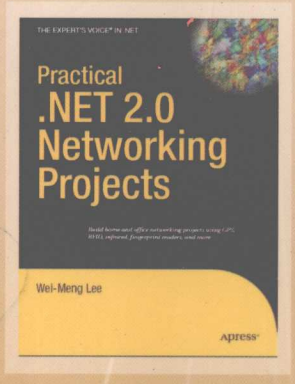
Practical .NET 2.0 Networking Projects

C#与VB.NET网络通信开发实战

[美] Wei-Meng Lee 著
田国法 吴兰陟 译

Microsoft MVP Most Valuable Professional

- 释放.NET网络通信技术的潜力
- 同时提供C#和VB.NET代码
- 涵盖各种通信应用：
  串行、红外、RFID、蓝牙、GPS……



人民邮电出版社
POSTS & TELECOM PRESS

TURING 图灵程序设计丛书 微软技术系列

Practical .NET 2.0 Networking Projects

C#与VB.NET网络通信开发实战

[美] Wei-Meng Lee 著
田国法 吴兰陟 译

人民邮电出版社
北 京

图书在版编目（CIP）数据

C#与 VB.NET 网络通信开发实战 /（美）李（Lee, W.M.）著；田国法，吴兰陟译．—北京：人民邮电出版社，2008.8
（图灵程序设计丛书）
ISBN 978-7-115-18196-1

Ⅰ．C… Ⅱ．①李…②田…③吴… Ⅲ．①C 语言—程序设计②BASIC 语言—程序设计 Ⅳ．TP312

中国版本图书馆 CIP 数据核字（2008）第 075951 号

内 容 提 要

本书阐述了如何使用 .NET 的一些关键网络通信技术，讨论了有线设备之间以及网络与无线设备之间的通信，并通过实例教会读者以简单直接的方式应用这些技术。书中从探讨理论背景开始，然后使用框架中的 API 创建各种网络应用程序，从蓝牙和 RFID 通信，到套接字编程和聊天服务。书中全部实例代码都同时提供 Visual Basic .NET 和 C # 版本。

本书适合各个层次的 .NET 开发人员阅读。

图灵程序设计丛书

C#与 VB.NET 网络通信开发实战

◆ 著 [美] Wei-Meng Lee
译 田国法 吴兰陟
责任编辑 陈兴璐

◆ 人民邮电出版社出版发行 北京市崇文区夕照寺街 14 号
邮编 100061 电子函件 315@ptpress.com.cn
网址 http://www.ptpress.com.cn
北京世纪雨田印刷有限公司印刷

◆ 开本：800×1000 1/16
印张：16.5
字数：390 千字 2008 年 8 月第 1 版
印数：1－4 000 册 2008 年 8 月北京第 1 次印刷
著作权合同登记号 图字：01-2007-4255 号
ISBN 978-7-115-18196-1/TP

定价：39.00 元

读者服务热线：(010)88593802 印装质量热线：(010)67129223
反盗版热线：(010)67171154

引　言

本书将阐述因.NET Framework 2.0而变得更加容易实现的一些关键网络技术，讨论了有线互联的机器之间以及网络与移动设备之间的通信。本书将通过示例项目以一种直接、简单易懂的方式来讲解这些技术。

本书共包含6章，各章分别介绍网络编程的一个特定方向。我们将使用.NET Framework中的API以及第三方SDK来构建各种先进的网络应用程序，覆盖从蓝牙和FRID通信到套接字编程与聊天服务器等内容的方方面面。书中将为每个项目构建可运行的实例，这些实例也可以被定制以适用于你自己的目的。这些精选的项目包括以下内容。

第 1 章：套接字编程

编写网络应用程序是程序设计中最有趣的领域之一。眼看着自己编写的程序成功地通过网络实现了通信，这是特别令人振奋的。在这一章里，我们将使用TCP/IP建立一个类似于Windows Live Messenger（或ICQ）的聊天程序。通过这个聊天程序，你将学会如何在.NET中进行网络编程，并了解建立多用户聊天程序时会遇到的种种挑战。

第 2 章：串行通信

串行通信是设备之间相互通信最古老的机制之一。从IBM PC及其兼容机开始，几乎所有的计算机都配备了一个或多个串行端口和一个并行端口。顾名思义，*串行端口*（serial port）每次1位地连续收发数据，而*并行端口*（parallel port）则使用8条独立的数据线每次收发8位数据。

尽管串行端口传输速度相对低于并行端口，但串行通信仍然是设备连接的流行选项，因为它简单而且成本较低。虽然当今的消费产品正在使用USB连接取代串行连接，但是仍有大量的设备把串行端口作为它们与外部世界连接的唯一途径。

这一章将介绍怎样使用.NET Framework 2.0和.NET Compact Framework 2.0里新的`SerialPort`类实现与其他串行设备的通信。我们将建立3个项目，用实例说明怎样使用串行通信。第一个项目是一个聊天程序，它允许（使用串行数据线或者蓝牙互相连接的）两台计算机进行通信。你可以以此程序为基础，对其进行扩展来实现与手机之类的其他外部串行设备的通信。你将学会如何通

过一个串行的蓝牙连接使用AT命令通过程序来控制你的移动电话。第二个项目是一个Pocket PC聊天程序，与前一个项目十分类似。第三个项目展示了怎样与GPS接收器通信，并从中提取有用的数据用于在地图上显示当前位置。

第 3 章：将指纹识别纳入.NET 程序

生物特征识别是确认个体身份最可靠的方式之一。现在，应该有很多人已经熟悉了微软指纹阅读器（Microsoft Fingerprint Reader）。使用微软指纹阅读器，你只需把手指放在读取器上，就能登录你的电脑。你也可以使用指纹阅读器提供的应用程序来为要求身份认证的网站保存用户ID和密码。随后你就可以将指纹作为钥匙，取出ID和密码，安全地登录那些站点。微软指纹读取器把需要为不同网站记住不同密码的烦恼一扫而尽。

在这一章，我们将说明如何使用GrFinger Fingerprint SDK将微软指纹阅读器集成到你的.NET 2.0 Windows应用程序中。我们将建立一个来访者鉴别系统，访问办公室的用户可以使用它在接待处登记。登记以后，下一次他再访问办公室时，只需简单地扫描一下指纹，系统就会记录他的来访。学校也可以改编这个程序用于考勤，比如在大型的阶梯教室等场合，考勤必须快速而有效地实施。

第 4 章：红外线编程

在流行的如WiFi（Wireless Fidelity，基于IEEE 802.11b标准的无线局域网）、蓝牙和其他无线技术的喧哗声中，有一种最简单而又最普遍的无线通信形式很容易被忽略——它就是红外线通信。其实只要用过遥控器就已经用过这种通信形式。红外线通信使用超出光谱中可见光红光的不可见波段。你可以在应用程序中利用它进行短程的、点到点的数据传输。因为使用了光，所以光路是使用红外线通信的必要条件。尽管有此限制，红外线还是在数码相机、PDA和笔记本电脑等设备中日益流行。

在这一章里，我们将讲解如何建立允许两台设备（及计算机）使用红外线进行无线通信的应用程序。你可以将这一章中阐述的编程技术改造并应用到其他编程任务中，如编写无线的网络游戏等。

第 5 章：RFID 编程

射频识别（Radio Frequency Identifications，RFID）是近来在IT行业大力宣传的一项技术。RFID系统是一种识别系统，通过无线电波从称为电子标签（tag）或应答器（transponder）的设备中接收数据。RFID应用在日常生活中随处可见——超市、图书馆、书店等。RFID提供了一种快速而有效的方式来收集信息，如仓库的库存盘点、物品下落的追踪等。

在这一章，我们将介绍如何构建利用RFID技术进行数据采集的Windows应用程序。我们将使

用2个RFID阅读器并分析它们各自的优缺点。

第6章：与外围设备交互

摄像头是当今大多数人都可以轻易购置的普通外围设备，经常用于视频会议。但是，除了视频会议以外，还可以用摄像头做些什么呢？对于.NET开发人员来说，答案非常多。而且你会高兴地看到，将摄像头集成到Windows应用程序中并不像想象中那么难。

除了将摄像头集成到应用程序以外，还可以将Windows应用程序连接到诸如传感器这样的外围设备，以监视周围环境的变化。

在这一章，我们通过将Windows程序与外围的传感器和摄像头接口，构建一个可以监视有害活动的安全系统。你可以探测到入侵者，并用摄像头记录入侵者的行动。

版 权 声 明

目　录

第1章 套接字编程

编写网络应用程序是程序设计中最有趣的领域之一。眼看着自己编写的程序成功地通过网络实现了通信，这是特别令人振奋的。在这一章里，我们将使用TCP/IP建立一个类似于Windows Live Messenger（或ICQ）的聊天程序。通过这个聊天程序，你将学会如何在.NET中进行网络编程，并了解建立多用户聊天程序时会遇到的种种挑战。

1.1 套接字编程介绍

套接字（socket）是网络计算机与应用程序之间发送和接收数据的方式的一种抽象描述。它描述了（可能在不同的计算机上，也可能在同一台计算机内的）两个通信点之间的连接。

在实际操作中，套接字编程往往与TCP/IP和UDP/IP通信相结合（关于TCP/IP和UDP/IP的更多信息参见下面的“理解IP、TCP和UDP”）。论及套接字编程时，以下3类信息是很重要的：

- 协议（如TCP/IP或UDP/IP）。
- IP地址（例如127.0.0.1）。
- 端口号（例如端口80）。

举例来说，对于http//www.apress.com这样的地址，你应该比较熟悉，这个地址用来指示Web浏览器加载位于www.apress.com的主页。http指定了使用的应用协议（HTTP使用TCP/IP传输数据），www.apress.com指定了地址（名称www.apress.com将会被DNS服务器解析成一个IP地址）。由于HTTP使用端口80进行通信，端口号80被隐式地指明，故没有出现在地址里。如图1-1所示，通信的双方都必须拥有IP地址。

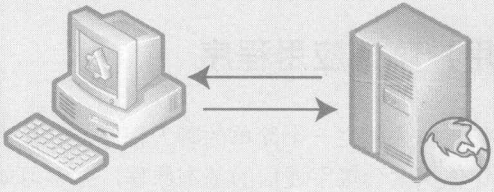


图1-1 Web浏览器和网络服务器之间的通信

尽管有TCP/IP这样的协议负责把数据从一个点传输到另一个点，但所传输数据的内容则需由诸如HTTP这样的应用协议来指定。

在.NET Framework里，套接字通信由Socket类来实现（该类位于System.Net.Sockets命名空间）。

理解IP、TCP和UDP

对于网络编程来说，深入地理解当下正在使用的一些常见网络协议是很重要的。首先是网际协议（Internet Protocol，IP）。IP指定了从一个点传送到另一个点的数据分组（如数据报datagrams）的格式和寻址方案。假设IP是一种邮递系统，你可以通过它把邮件从一个地方寄到另一个地方。你只需写上收件者地址并把邮件丢进邮箱里。随后邮局会试图把邮件投递给收件人。但是，你不能确定你的邮件肯定会到达目的地，也不会知道它究竟何时到达。

为了确保邮件被正确地投递，你必须使用额外的服务，比如挂号信。与上述情形类似，我们需要将其他协议与IP联合使用，以保证数据分组传送无误。传输控制协议（Transmission Control Protocol，TCP）正是这样一种协议。TCP是一种面向连接的网络协议，它（通过应答机制）保证数据分组可靠并有序地传送。作为流行的网络协议，与IP协同工作的TCP已被Web浏览器和电子邮件客户这样的应用程序广泛采用。

TCP确保了传送的正确性，但它也有不便的地方。正如要花更多的钱来寄送的挂号信一样，TCP给被发送的数据分组加上了额外的报头，增大了分组的尺寸。因此，开发人员有时会将用户数据报协议（User Datagram Protocol，UDP）与IP联用。UDP是一种无连接的网络协议，同样把数据分组从一点发送到另一个点，只有一个例外——它并不提供可靠的、有保障的传送。由于UDP不对传送提供保障，数据分组将包含更多的有效信息并能更快地传送。使用UDP的开发人员必须建立自己的逻辑以确保数据分组的正确传送。这也与邮寄的例子很相似：你可以自己给收件人打电话，看他们是否已经收到你寄的邮件。如果他们没有收到，你可能需要重新寄。对于那些传送小数据分组且不需要数据精确组装的应用程序来说，UDP是非常有用的。这类程序包括简单文件传输协议（Trivial File Transfer Protocol，TFTP）、域名系统（Domain Name System，DNS）以及语音IP（Voice over IP，VoIP）。

1.2　创建自己的多用户聊天应用程序

在本章的这一部分，我们将首先建立一个简单的聊天程序，它允许连接到中央服务器的任何人互相进行通信。这样可以让你探索套接字通信的基本原理，并学习如何向所有已连接的用户广播消息。

图1-2展示了本章的这一部分将要的建立的应用程序。

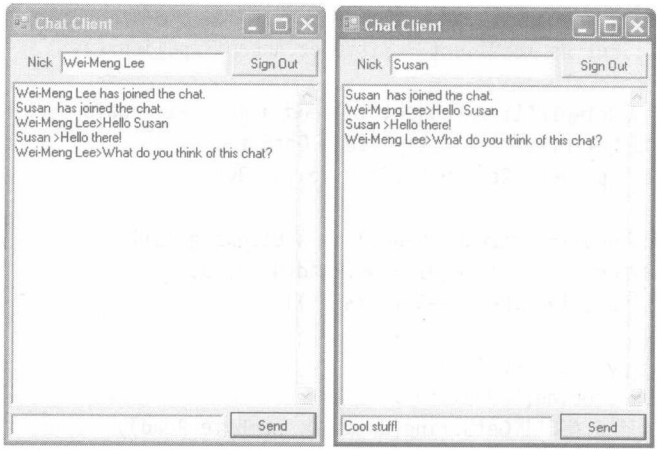


图1-2 即将创建的聊天程序

1.2.1 为网络通信使用 TcpClient 和 TcpListener 类

创建聊天程序通常涉及套接字编程——创建一个客户和服务器之间的连接，使客户和服务器都能发送和接收消息。System.Net.Sockets命名空间提供套接字编程所需的功能。在这个项目我们将使用System.Net.Sockets命名空间里的2个类：TcpClient和TcpListener。

TcpClient类实现了使用TCP发送和接收数据的套接字。因为与远程设备的连接被表示为流，数据可以使用.NET Framework的流处理技术来读取和写入。

TcpListener类以阻塞同步模式提供用于监听和接受外来连接请求的简单方法。

下面的示例代码实现了一个简单的等待外来连接的服务器（一个控制台应用程序）。

Visual Basic 2005

```
Imports System.Net.Sockets
Imports System.Text

Module Module1
    '---port number to use for listening---
    Const portNo As Integer = 500

    Sub Main()
        Dim localAdd As System.Net.IPAddress = _
            System.Net.IPAddress.Parse("127.0.0.1")

        '---listen at the local address---
        Dim listener As New TcpListener(localAdd, portNo)
        listener.Start()
```

```
        '---Accepts a pending connection request---
        Dim tcpClient As TcpClient = listener.AcceptTcpClient()

        '---use a NetworkStream object to send and receive data---
        Dim ns As NetworkStream = tcpClient.GetStream
        Dim data(tcpClient.ReceiveBufferSize) As Byte

        '---read incoming stream; Read() is a blocking call---
        Dim numBytesRead As Integer = ns.Read(data, 0, _
            CInt(tcpClient.ReceiveBufferSize))

        '---display data received---
        Console.WriteLine("Received :" & _
            Encoding.ASCII.GetString(data, 0, numBytesRead))

        '---prevent the console window from closing immediately---
        Console.ReadLine()
    End Sub

End Module
```

C# 2005

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Net.Sockets;

namespace Server_CS
{
    class Program
    {
        //---port number to use for listening---
        const int portNo = 500;
        static void Main(string[] args)
        {
            System.Net.IPAddress localAdd =
               System.Net.IPAddress.Parse("127.0.0.1");

            //---listen at the local address---
            TcpListener listener = new TcpListener(localAdd, portNo);
            listener.Start();

            //---Accepts a pending connection request---
            TcpClient tcpClient = listener.AcceptTcpClient();
```

```
            //---use a NetworkStream object to send and receive
            // data---
            NetworkStream ns = tcpClient.GetStream();
            byte[] data = new byte[tcpClient.ReceiveBufferSize];

            //---read incoming stream; Read() is a blocking call---
            int numBytesRead = ns.Read(data, 0,
               System.Convert.ToInt32(tcpClient.ReceiveBufferSize));

            //---display data received---
            Console.WriteLine("Received :" +
               Encoding.ASCII.GetString(data, 0, numBytesRead));

            //---prevent the console window from closing
            // immediately---
            Console.ReadLine();
        }
    }
}
```

要连接到服务器并向它发送一个字符串，客户代码（一个控制台应用程序）将是下面这样的：

Visual Basic 2005

```
Imports System.Net.Sockets
Imports System.Text

Module Module1
    Const portNo As Integer = 500
    Sub Main()
        Dim tcpclient As New TcpClient

        '---connect to the server---
        tcpclient.Connect("127.0.0.1", portNo)

        '---use a NetworkStream object to send and receive data---
        Dim ns As NetworkStream = tcpclient.GetStream
        Dim data As Byte() = Encoding.ASCII.GetBytes("Hello")

        '---send the text---
        ns.Write(data, 0, data.Length)
    End Sub
End Module
```

C# 2005

```
using System;
```

```
using System.Collections.Generic;
using System.Text;
using System.Net.Sockets;

namespace Client_CS
{
    class Program
    {
        const int portNo = 500;
        static void Main(string[] args)
        {
            TcpClient tcpclient = new TcpClient();

            //---connect to the server---
            tcpclient.Connect("127.0.0.1", portNo);

            //---use a NetworkStream object to send and receive
            // data---
            NetworkStream ns = tcpclient.GetStream();
            byte[] data = Encoding.ASCII.GetBytes("Hello");

            //---send the text---
            ns.Write(data, 0, data.Length);
        }
    }
}
```

注意，NetworkStream对象操作字节数组，因而需要使用来自System.Text命名空间的Encoding.ASCII.GetString()和Encoding.ASCII.GetBytes()方法来将字节数组转换成字符串，反之亦然。

上面的例子是比较简单的——它包含了服务器代码和客户代码。服务器在127.0.0.1使用端口500打开一个套接字并监听外来TCP连接。当连接建立起来以后，由一个NetworkStream对象读取客户发来的数据。到达的数据随后显示在控制台上。另一方面，客户在127.0.0.1打开一个连接，然后使用NetworkStream对象向服务器发送一个字符串。

但是，当服务器需要同时与多个客户通信并能同时发送和接收消息时，问题就会变得复杂得多。为了实现这些，必须满足以下几条。

- 服务器必须能够与多个客户建立连接。
- 服务器必须能够从客户异步读取数据并能在任何时刻向客户发送消息。
- 客户必须能够从服务器异步读取数据并能在任何时刻向服务器发送消息。

接下来的几节将解决这3个问题。

1.2.2 构建服务器

聊天程序有两个部件——服务器和客户，我们来首先构建服务器。用Vistual Studio 2005创建一个控制台程序项目，将该项目命名为**Server**。

在默认的Module1.vb/Program.cs文件里，首先导入System.Net.Socket命名空间，它包含这个项目将要用到的所有相关的类。

Visual Basic 2005

```
Imports System.Net.Sockets
```

C# 2005

```
using System.Net.Sockets;
```

接下来，声明一个常量来存储这个应用程序使用的端口号。对这个程序，我们使用端口号500。

Visual Basic 2005

```
Module Module1
    Const portNo As Integer = 500
```

C# 2005

```
class Program
{
    const int portNo = 500;
```

提示 如果你在服务器（或者客户）上安装了防火墙，请确保打开端口500，以便这个应用程序运行。

我们还需要定义所要监听的本地地址，然后创建一个TcpListener类的实例，用来监听来自TCP客户的连接。

Visual Basic 2005

```
Sub Main()
    Dim localAdd As System.Net.IPAddress = _
       System.Net.IPAddress.Parse("127.0.0.1")
    Dim listener As New TcpListener(localAdd, portNo)
```

C# 2005

```
static void Main(string[] args)
{
    System.Net.IPAddress localAdd =
       System.Net.IPAddress.Parse("127.0.0.1");
    TcpListener listener = new TcpListener(localAdd, portNo);
```

在Main()函数里，使用来自类TcpListener的Strart()方法来开始监听外来连接请求。AcceptTcpClient()方法是一个阻塞式的调用，直到连接建立起来以后程序才会继续执行。因为这个示例里的服务器需要同时为多个客户提供服务，我们将为每一个用户创建一个ChatClient类（稍后将定义）的实例。服务器将会无限地循环检查，若有客户请求连接，则接受。

Visual Basic 2005

```
Sub Main()
    Dim localAdd As System.Net.IPAddress = _
       System.Net.IPAddress.Parse("127.0.0.1")
    Dim listener As New TcpListener(localAdd, portNo)
    listener.Start()
    While True
        Dim user As New ChatClient(listener.AcceptTcpClient)
    End While
End Sub
```

C# 2005

```
static void Main(string[] args)
{
    System.Net.IPAddress localAdd =
       System.Net.IPAddress.Parse("127.0.0.1");
    TcpListener listener = new TcpListener(localAdd, portNo);
     listener.Start();
     while (true)
     {
         ChatClient user = new
            ChatClient(listener.AcceptTcpClient());
     }
}
```

完整的Module1.vb源文件如下所示。

Visual Basic 2005

```
Imports System.Net.Sockets

Module Module1
    Const portNo As Integer = 500
    Sub Main()
        Dim localAdd As System.Net.IPAddress = _
           System.Net.IPAddress.Parse("127.0.0.1")
        Dim listener As New TcpListener(localAdd, portNo)
        listener.Start()
        While True
```