第 5 章

Java 网络编程

技能目标

- 理解 IP 地址的组成和分类
- 掌握 InetAddress 类中方法的使用
- 会实现基于 TCP 协议的 Socket 编程
- 了解基于 UDP 协议的 Socket 编程
- 了解 JUnit 测试框架的搭建

本章任务

学习本章，需要完成以下 4 个工作任务。记录学习过程中遇到的问题，可以通过自己的努力或访问 kgc.cn 解决。

任务 1：查看 IP 地址

任务 2：实现基于 TCP 协议的 Socket 编程

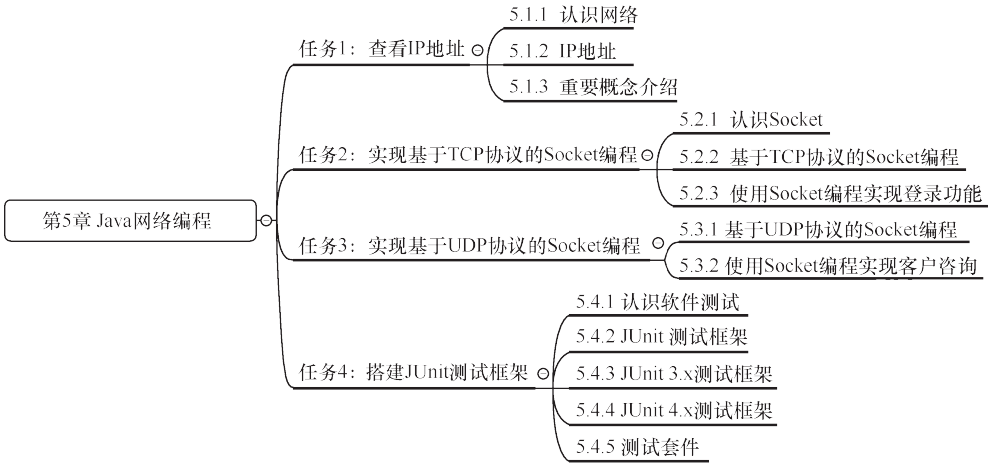
使用基于 TCP 协议的 Socket 编程模拟搭建客户端以及服务器端。

任务 3：实现基于 UDP 协议的 Socket 编程

本任务模拟在线客服系统的客户咨询，并实现客户端与服务器端之间的信息交流。

任务 4：搭建 JUnit 测试框架

在本机搭建 JUnit 3.x 测试框架并实现测试断言



任务 1 查看 IP 地址

关键步骤如下。

- 打开命令提示窗口。
- 输入命令查看本机的 IP 地址。

5.1.1 认识网络

当今社会，大家对“网上购物”“网络支付”“移动互联网”“互联网 +”等名词一定不再陌生，因为互联网技术已经渗透到生活的方方面面：从手写的书信到快捷的电子邮件，从面对面交易到足不出户的网上交易，从圆桌会议到视频会谈，从烦琐的资料查找到便捷的搜索……如果没有了计算机网络，世界将陷入瘫痪。

的确，人们生活在一个便捷的“网络时代”。但是，网络到底是什么？它是如何组建并为人们提供服务的呢？这一章将回答这些问题。下面，让我们赶快进入网络的精彩世界吧！

1. **网络的概念和分类**

简单来说，网络就是连接在一起共享数据和资源的一组计算机。

分布在不同地理区域的计算机与专门的外部设备通过通信线路互连在一起，形成一个规模大、功能强的网络系统，从而使众多的计算机可以方便地互相传递信息、共享信息资源，如图 5.1 所示。

计算机网络旨在实现数据通信。数据可以有多种形式，如文本、图片或声音。进行数据通信的两台计算机可以相距很近（如同一间办公室），也可以在地理位置上相隔很远（如不同的国家）。

按照地理覆盖范围，计算机网络可以划分为局域网、城域网和广域网。

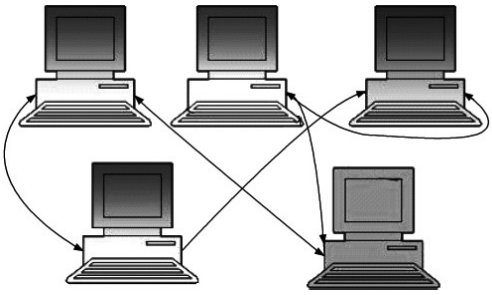


图 5.1　计算机网络

（1）局域网

局域网（LAN）局限在小的地理区域内或单独的建筑物内，被用于连接公司办公室、实验室或工厂里的个人计算机和工作站。如图 5.2 所示为一个办公室局域网。

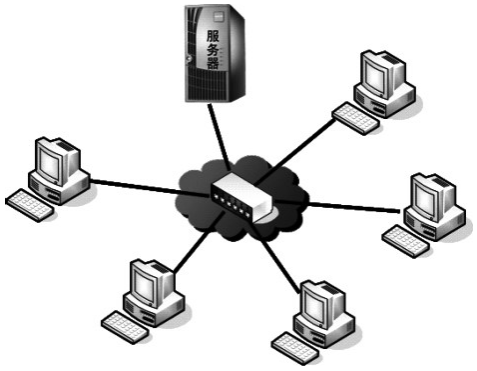


图 5.2　局域网

（2）城域网

城域网（MAN）覆盖城市或城镇内的广大地理区域，是在一个城市范围内所建立的计算机通信网。例如，一个大学不同校区的计算机相互连接，以及位于不同区域的银行机构提供的相互间的网络通信都形成了一个城域网。如图 5.3 所示为一个教育城域网。

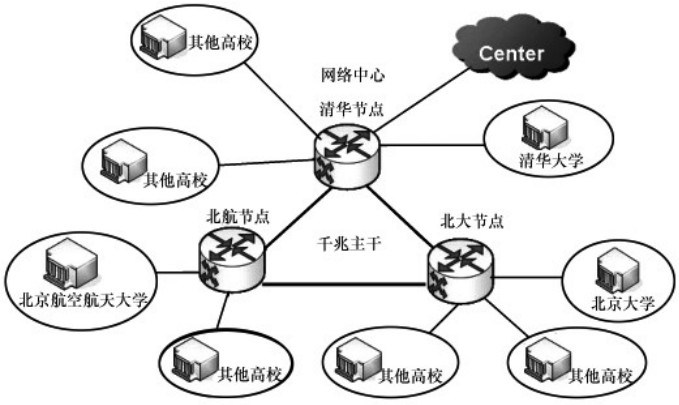


图 5.3　城域网

（3）广域网

广域网（WAN）是在一个更广泛的地理范围内所建立的计算机通信网，其范围可以超越城市和国家以至全球，因而对通信的要求及复杂性都比较高。

如图 5.4 所示，网络将多个终端系统（如 A、C、H 和 K）和多个中间系统（如 B、D、E、F、G、I 和 J）连接起来，通过该网络可实现终端系统间的数据通信。这些终端系统可以是单个计算机或局域网。

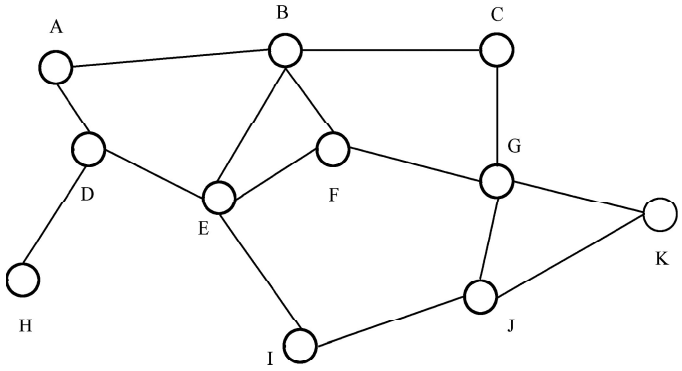


图 5.4　广域网

2. **网络分层模型**

如同一个公司的组织架构一样，网络上的信息传递，也是由不同的层级负责不同的工作任务。但由于各个计算机厂商都采用私有的网络模型，这会给通信带来诸多麻烦，国际标准化组织（International Standard Organization，ISO）于 1984 年颁布了开放系统互连（Open System Interconnection，OSI）参考模型。OSI 参考模型是一个开放式体系结构，它规定将网络分为 7 层，每一层在网络信息传递中都发挥不同的作用。表 5-1 列举了 OSI 模型中每层的功能。

表5-1　OSI参考模型

分　层	功　能
应用层	网络服务和最终用户的接口
表示层	数据的表示、安全和压缩
会话层	建立、管理和终止会话
传输层	定义传输数据的协议端口号，流量控制和差错恢复
网络层	进行逻辑地址寻址，实现不同网络之间的路径选择
数据链路层	建立逻辑连接，进行硬件地址寻址，差错校验等功能
物理层	建立、维护、断开物理连接

另外一个著名的模型是 TCP/IP 模型。TCP/IP 是传输控制协议 / 网络互联协议（Transmission Control Protocol/Internet Protocol）的简称。早期的 TCP/IP 模型是 4 层结构。

在后来的使用过程中，借鉴 OSI 的 7 层参考模型，将网络接口层划分为物理层和数据链路层，形成新的 5 层结构。TCP/IP 模型前 4 层与 OSI 参考模型的前 4 层相对应，其功能也非常类似，而应用层则与 OSI 参考模型的最高 3 层相对应，如图 5.5 所示。

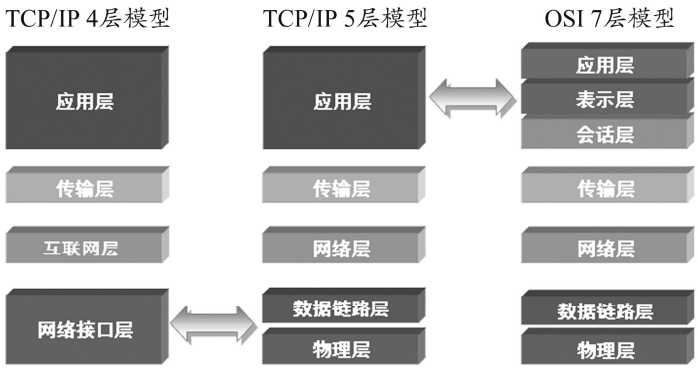


图 5.5　OSI 参考模型与 TCP/IP 模型

5.1.2　IP 地址

1．IP 地址概述

网络如此之庞大，要将如此众多的计算机互连，使信息获得共享，如何在网络中找到目标计算机呢？下面以信件邮寄的过程为例来分析这个问题。


IP 地址的组成

首先要知道对方的地址，然后在信封上写明收件人的地址，邮递员就能根据地址将信件正确地送到对方手中。另外，在邮件末尾写明发件人的地址，对方就可以根据地址回信。可见，地址是双方联系的关键要素。

类似地，要实现两台计算机之间的通信，双方都要具有地址。在网络中使用一种具有层次结构的逻辑地址来标识一台主机，这个地址称为 IP 地址。IP 地址用来唯一标识网络中的每一台计算机。

IP 地址目前存在 IPv4 和 IPv6 两种标准。

2．IP 地址的组成和分类

（1）IP 地址的组成

IPv4 地址有 32 位，由 4 个 8 位的二进制数组成，每 8 位之间用圆点隔开，如 11000000.10101000.00000010.00010100。由于二进制不便记忆且可读性较差，所以通常都把二进制数转换成十进制数表示，如 196.168.2.20。因此，一个 IP 地址通常由用 3 个点号分开的 4 个十进制数表示，称为点分十进制。

IPv6 地址有 128 位，由 8 个 16 位的无符号整数组成，每个整数用 4 个十六进制数表示，这些数之间用冒号（:）分开。例如：3ffe:3201:1401:1280:c8ff:fe4d:db39:1988。

（2）IP 地址的分类

IP 地址包含网络地址和主机地址两部分。其中，网络地址决定了可以分配的最大

网络数，主机地址决定了一个网络中可以存在的计算机的最大数量。

IP 地址的网络地址由互联网数字分配机构（The Internet Assigned Numbers Authority，IANA）统一分配，以保证 IP 地址的唯一性。IANA 将 IP 地址分为 A、B、C、D、E 共五类，并规定每个类别网络地址和主机地址的长度，如图 5.6 所示。

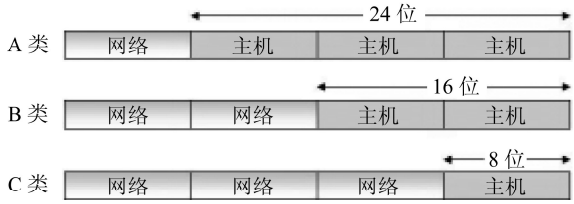


图 5.6　IP 地址分类

A 类 IP 地址：第一组数字表示网络地址，其余三位表示主机地址。A 类地址的第一个十进制数的有效取值范围为 1 ～ 126。

B 类 IP 地址：前两组数字表示网络地址，其余两位表示主机地址。B 类地址的第一个十进制数的有效取值范围为 128 ～ 191。

C 类 IP 地址：前三组数字表示网络地址，其余一位表示主机地址。C 类地址的第一个十进制数的有效取值范围为 192 ～ 223。

D 类 IP 地址：不分网络地址和主机地址，用于组播通信，不能在互联网上作为节点地址使用。D 类地址的第一个十进制数的有效取值范围为 224 ～ 239。

E 类 IP 地址：不分网络地址和主机地址，用于科学研究，也不能在互联网上作为节点地址使用。E 类地址的第一个十进制数的有效取值范围为 240 ～ 254。

除此之外，还有一些特殊的 IP 地址，例如：

0.0.0.0：表示本机。

127.0.0.1：表示本机回环地址，通常用在本机上 ping 此地址来检查 TCP/IP 协议安装是否正确（ping 命令将在后面进行说明）。

255.255.255.255：表示当前子网，一般用于向当前子网广播信息。

3. IP 地址的配置和检测

为了使一台计算机接入网络，除了必备的网络设备之外，还要进行相应的 TCP/IP 设置。

可按如下步骤配置 TCP/IP。

（1）打开“控制面板”窗口，双击“网络连接”图标。

（2）双击“本地连接”图标，打开“本地连接 状态”对话框，如图 5.7 所示。

（3）单击“属性”按钮，打开“本地连接 属性”对话框，如图 5.8 所示。

（4）勾选“Internet 协议版本 4（TCP/IPv4）”复选框并单击“属性”按钮，打开“Internet 协议版本 4（TCP/IPv4）属性”对话框，如图 5.9 所示。

（5）选中“使用下面的 IP 地址”单选按钮并输入 IP 地址、子网掩码和默认网关（你的网络中连接到其他网络的计算机或路由器），如图 5.9 所示。

图 5.7 “本地连接状态”对话框

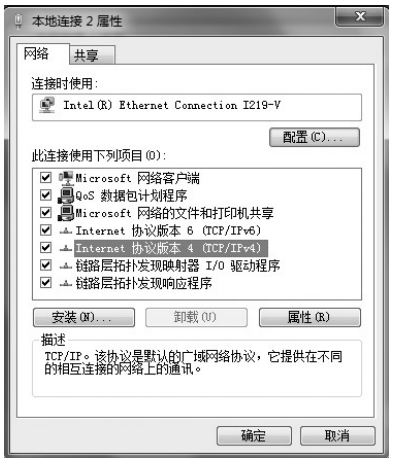


图 5.8 “本地连接属性”对话框

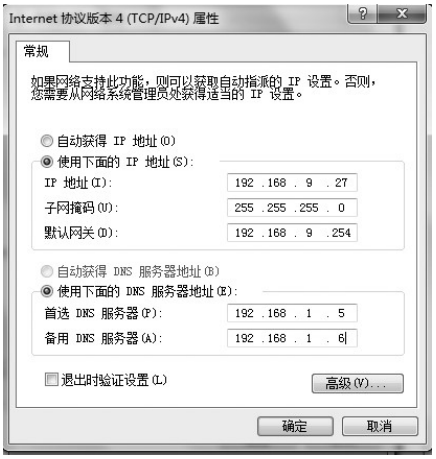


图 5.9 “TCP/IP 协议版本 4（TCP/IPv4）属性”对话框

（6）选中“使用下面的 DNS 服务器地址”单选按钮并输入 DNS 地址，如图 5.9 所示。

（7）单击“确定”按钮完成设置。

注意

实际应用中，在配置局域网（支持 DHCP 服务）中计算机的 IP 地址时，为了避免人为输入产生地址冲突的错误，通常选中“自动获得 IP 地址”单选按钮。

设置了 IP 地址之后，可能出现网络连接不通的故障，怎么检测呢？这就需要使用几个经典的 DOS 命令了。

首先，使用 ipconfig 命令查看本机的 IP 地址、子网掩码、默认网关等信息，判断 TCP/IP 属性是否设置正确，如图 5.10 所示。

然后，使用 ping 命令测试网络是否通畅，检测故障原因。

ping 命令的语法格式如下。

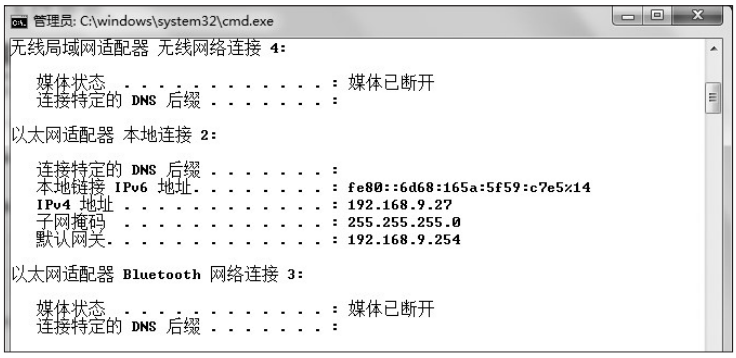


图 5.10　ipconfig 命令的输出结果

ping 目标 IP 地址

例如，ping 本机回环地址，检测 IP 设置是否正确，如图 5.11 所示。

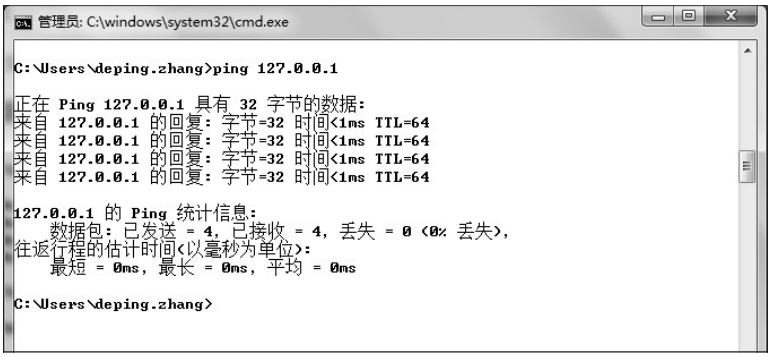


图 5.11　ping 本机回环地址的输出结果

另外，可以 ping 默认网关的 IP 地址来检验连接是否通畅，ping 某一远程计算机来测试是否可以与远程主机正常通信。

最后，根据检查结果排除故障。例如，修改 IP 地址，检查网线、网络适配器（简称网卡）是否松动或接触不良等。

至此，任务 1 已经全部完成，自己动手试试看吧。

5.1.3　重要概念介绍

完成了任务 1，对于网络编程的基础知识掌握还不够，还需要了解以下几个重要概念。

1. 端口

网络中的一台计算机通常可以使用多个进程同时提供网络服务。因此除了 IP 地址，每台主机还有若干个端口号，用于在收发数据时区分该数据发给哪个进程或者是从哪个进程发出的。端口是计算机与外界通信的入口和出口，它是一个 16 位的整数，范围是

0 ～ 65535（2^{16}-1）。在同一台主机上，任何两个进程不能同时使用同一个端口。

2. 域名与 DNS 域名解析

前面已经提到，IP 地址用来唯一定位一台计算机，也就是说只有通过 IP 地址才能找到一台网络中的主机。那么，为什么在上网时轻松地输入网址，就能够获得这个远程的 Web 服务器提供的资源呢？例如，为什么在浏览器的地址栏中输入 www.taobao.com 就能进入“淘宝”网站呢？难道它没有 IP 地址吗？

答案当然是否定的。人们希望记忆名字而不是枯燥的数字，因此就需要一个系统将一个名称映射为它的 IP 地址。DNS（Domain Name System，域名系统）被广泛使用，用于将域名（如 taobao.com）映射成 IP 地址。

DNS 服务器是如何解析域名的呢？如图 5.12 所示，在浏览器中输入域名 www.taobao.com，主机在向 www.taobao.com 发出请求之前要先知道它的 IP 地址。主机会调用域名解析程序，向设定的 DNS 服务器发送信息，请求获得 www.taobao.com 的 IP 地址，如果本地 DNS 服务器没有存储相应的信息，它会发送信息到根 DNS 服务器获得 .com DNS 服务器的 IP，然后向 .com DNS 服务器发送查询请求获得 taobao.com DNS 服务器的 IP 地址，最终获得 www.taobao.com 的 IP 地址。

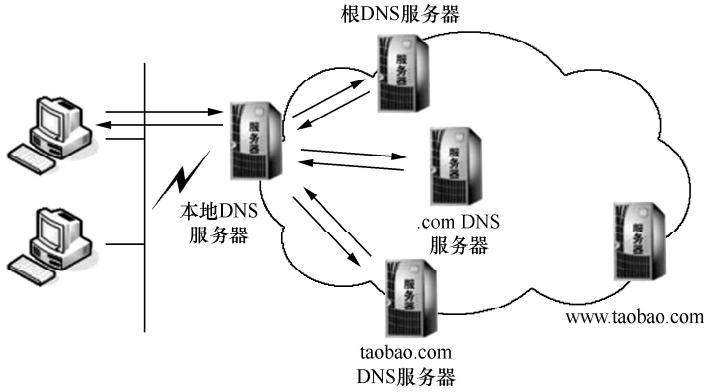


图 5.12　DNS 域名解析过程

3. 网络服务器

网络服务器通常是指在网络环境下，具有较高计算能力，能够为用户提供特殊服务功能的计算机，下面简单介绍目前常用的几种网络服务器。

（1）邮件服务器

邮件服务器是一种用来负责电子邮件收发管理的设备。邮件服务器构成了电子邮件系统的核心，负责网络中电子邮件的定位和收发管理工作。它的工作遵循一定的工作协议，通过对这些协议的遵守，世界各地的邮件服务器才能统一工作，共同管理网络中庞大的电子邮件的传送。

（2）Web 服务器

Web 服务器也称为 WWW 服务器，主要功能是提供网上信息浏览服务。Web 服务器不仅能够存储信息，还能够通过 Web 浏览器为用户在提供信息的基础上运行脚本和

程序。下面介绍几种常用的 Web 服务器。

1）Microsoft IIS

Microsoft 的 Web 服务器产品为 Internet Information Server（IIS），IIS 是允许在公共 Intranet 或 Internet 上发布信息的 Web 服务器。IIS 是目前最流行的 Web 服务器产品之一，很多著名的网站都是建立在 IIS 的平台上。IIS 提供了一个图形界面的管理工具，称为 Internet 服务管理器，可用于监视配置和控制 Internet 服务。

2）Apache 服务器

Apache 仍然是世界上用得最多的 Web 服务器。它的成功之处主要在于它的源代码开放、有一支开放的开发队伍、支持跨平台的应用（可以运行在绝大多数的 UNIX、Windows、Linux 操作系统平台上）以及它的可移植性等方面。

3）Tomcat 服务器

Tomcat 是一个开放源代码、运行 Servlet 和 JSP Web 应用软件的基于 Java 的 Web 应用服务器。Tomcat Server 是根据 Servlet 和 JSP 规范执行的，因此可以说 Tomcat Server 也遵循了 Apache-Jakarta 规范且比绝大多数商业应用软件服务器要好用。Tomcat 是基于 Apache 许可证下开发的自由软件，因此目前许多 Web 服务器都采用 Tomcat。

另外，还有原 IBM 的 WebSphere、BEA 的 WebLogic 等，也是市面上比较常见的 Web 服务器。

4. 网络通信协议

网络通信协议是为了在网络中不同的计算机之间进行通信而建立的规则、标准或约定的集合。它规定了网络通信时，数据必须采用的格式以及这些格式的意义。就好像人们在交谈时约定都使用英语或者都使用普通话一样。在网络编程时，常用的网络协议有以下几种。

（1）TCP/IP 协议族

TCP/IP 是 Transmission Control Protocol/Internet Protocol 的简称。它是用于计算机网络通信的协议集，即协议族。该协议族是 Internet 最基本的协议，它不依赖于任何特定的计算机硬件或操作系统，提供开放的协议标准。目前，绝大多数的网络操作系统都提供对 TCP/IP 协议族的支持，它已经成为 Internet 的标准协议。TCP/IP 协议族包括 IP 协议、TCP 协议、UDP 协议和 ARP 协议等诸多协议，其核心协议是 IP 协议和 TCP 协议，所以有时也将 TCP/IP 协议族简称为 TCP/IP 协议。

（2）TCP 协议

TCP 是 Transmission Control Protocol 的简称，中文名称为传输控制协议。TCP 是一种面向连接的、可靠的、基于字节流的传输层通信协议。TCP 要求通信双方必须建立连接之后才开始通信，通信双方可以同时进行数据传输，它是全双工的，从而保证了数据的正确传送。

（3）UDP 协议

UDP 是 User Datagram Protocol 的简称，中文名称为用户数据报协议。UDP 协议是一个无连接协议，在传输数据之前，客户端和服务器并不建立和维护连接。UDP 协议

的主要作用是把网络通信的数据压缩为数据报的形式。

任务 2　实现基于 TCP 协议的 Socket 编程

关键步骤如下。

- 两个端点进行连接。
- 打开传递信息的输入 / 输出流。
- 传递数据、接收数据。
- 关闭连接。

5.2.1　认识 Socket


Socket 简介

1．Socket 概述

Java 最初是作为网络编程语言出现的，它对网络的高度支持，使得客户端和服务器端流畅的沟通变成现实。而在网络编程中，使用最多的就是 Socket，每一个实用的网络程序都少不了它的参与。那么到底什么是 Socket 呢？

在计算机网络编程技术中，两个进程或者说两台计算机可以通过一个网络通信连接实现数据的交换，这种通信链路的端点就被称为“套接字”（英文名称也就是 Socket），Socket 是网络驱动层提供给应用程序的一个接口或者说一种机制。举一个物流送快递的例子来说明 Socket，发件人将写有收货人地址信息的货物送到快递站，发件人不用关心物流是如何进行的，货物被送到收货人所在地区的快递站点，进行配送，收货人等待收货就可以了，这个过程很形象地说明了信息在网络中传递的过程。其中，货物就是数据信息，2 个快递站点就是 2 个端点 Socket。信息如何在网络中寻址传递，应用程序并不用关心，只负责准备发送数据和接收数据即可。

2．Socket 通信原理

对于编程人员来说，无须了解 Socket 底层机制是如何传送数据的，而是直接将数据提交给 Socket，Socket 会根据应用程序提供的相关信息，通过一系列计算，绑定 IP 及信息数据，将数据交给驱动程序向网络上发送。如图 5.13 所示是 Socket 通信过程。

Socket 的底层机制非常复杂，Java 平台提供了一些虽然简单但是相当强大的类，可以更简单有效地使用 Socket 开发通信程序而无须了解底层机制。

3．java.net 包

java.net 包提供了若干支持基于套接字的客户端 / 服务器通信的类。

java.net 包中常用的类有 Socket、ServerSocket、DatagramPacket、DatagramSocket、InetAddress、URL、URLConnection 和 URLEncoder 等。

为了监听客户端的连接请求，可以使用 ServerSocket 类。Socket 类实现用于网络上进程间通信的套接字。DatagramSocket 类使用 UDP 协议实现客户端和服务器套接字。

DatagramPacket 类使用 DatagramSocket 类的对象封装设置和收到的数据报。InetAddress 类表示 Internet 地址。在创建数据报报文和 Socket 对象时，可以使用 InetAddress 类。接下来将详细阐述这些类的使用。

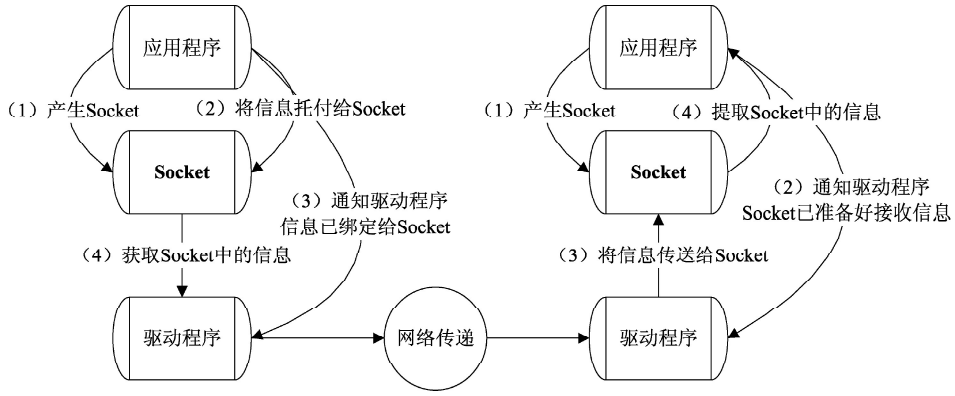


图 5.13　Socket 通信原理

5.2.2　基于 TCP 协议的 Socket 编程

java.net 包的两个类 Socket 和 ServerSocket，分别用来实现双向安全连接的客户端和服务器端，它们是基于 TCP 协议进行工作的，它的工作过程如同打电话的过程，只有双方都接通了，才能开始通话。

进行网络通信时，Socket 需要借助数据流来完成数据的传递工作。如果一个应用程序要通过网络向另一个应用程序发送数据，只要简单地创建 Socket，然后将数据写入到与该 Socket 关联的输出流即可。对应的，接收方的应用程序创建 Socket，从相关联的输入流读取数据即可。Socket 通信模型如图 5.14 所示。

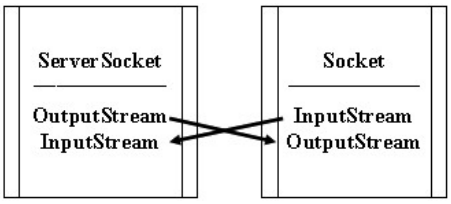


图 5.14　Socket 通信模型

注意

2 个端点在基于 TCP 协议的 Socket 编程中，经常一个作为客户端，一个作为服务器端，也就是遵循 client-server 模型。

1. Socket 类

Socket对象在客户端和服务器之间建立连接。可用Socket类的构造方法创建套接字，并将此套接字连接至指定的主机和端口。以下是与此 Socket 对象关联的构造方法和一些常用方法。

（1）构造方法

第一种构造方法以主机名和端口号作为参数来创建一个 Socket 对象。创建 Socket 对象时可能抛出 UnknownHostException 或 IOException 异常，必须捕获它们。

```
Socket s=new Socket (hostName, port);
```

另一种构造方法以 InetAddress 对象和端口号作为参数来创建一个 Socket 对象。构造方法可能抛出 IOException 或 UnknownHostException 异常，必须捕获并处理它们。

```
Socket s=new Socket(address, port);
```

（2）常用方法

Socket 类的常用方法如表 5-2 所示。

表5-2　Socket类的常用方法

方　法	说　明
InetAddress getInetAddress()	返回与Socket对象关联的InetAddress
int getPort()	返回此Socket对象所连接的远程端口
int getLocalPort()	返回此Socket对象所连接的本地端口
InputStream getInputStream()	返回与此套接字关联的InputStream
OutputStream getOutputStream()	返回与此套接字关联的OutputStream
void close()	关闭该Socket

2. ServerSocket 类

ServerSocket 对象等待客户端建立连接，连接建立以后进行通信。

（1）构造方法

可用的构造方法有两种。第一种接受端口号作为参数创建 ServerSocket 对象，创建此对象时可能抛出 IOException 异常，必须捕获和处理它。

```
ServerSocket ss=new ServerSocket(port);
```

另一种接受端口号和最大队列长度作为参数，队列长度表示系统在拒绝连接前可以拥有的最大客户端连接数。

```
ServerSocket ss=new ServerSocket(port, maxqu);
```

（2）常用方法

Socket 类中列出的常用方法也适用于 ServerSocket 类。此外，ServerSocket 类具有 accept() 方法，此方法用于等待客户端发起通信，这样 Socket 对象就可用于进一步的数据传输。

5.2.3 使用 Socket 编程实现登录功能

1. 实现单用户登录

Socket 网络编程一般分成如下 4 个步骤进行。

（1）建立连接。

（2）打开 Socket 关联的输入 / 输出流。

（3）从数据流中写入信息和读取信息。

（4）关闭所有的数据流和 Socket。

接下来，就使用这两个类模拟实现用户登录的功能，实现客户端向服务器端发送用户登录信息，服务器端显示这些信息。

示例 1

模拟用户登录的功能，实现客户端发送用户登录信息，服务器端显示登录信息并响应给客户端登录成功。

客户端实现步骤如下。

1）建立连接，连接指向服务器及端口。

2）打开 Socket 关联的输入 / 输出流。

3）向输出流中写入信息。

4）从输入流中读取响应信息。

5）关闭所有的数据流和 Socket。

客户端关键代码：

```
// 建立客户端 Socket 连接，指定服务器的位置为本机以及端口为 8800
Socket socket=new Socket("localhost",8800);
// 打开输入 / 输出流
OutputStream os=socket.getOutputStream();
InputStream is=socket.getInputStream();
// 发送客户端登录信息，即向输出流写入信息
String info=" 用户名：Tom; 用户密码：123456";
os.write(info.getBytes());
socket.shutdownOutput();
// 接收服务器端的响应，即从输入流中读取信息
String reply=null;
BufferedReader br=new BufferedReader(new InputStreamReader(is));
while(!((reply=br.readLine())==null)){
    System.out.println(" 我是客户端，服务器的响应为："+reply);
}
// 关闭资源
……
```

服务器端实现步骤如下。

1）建立连接，监听端口。

2）使用 accept() 方法等待客户端发起通信

3）打开 Socket 关联的输入 / 输出流。

4）向输出流中写入信息。

5）从输入流中读取响应信息。

6）关闭所有的数据流和 Socket。

服务器端关键代码：

```
// 建立一个服务器 Socket（ServerSocket），指定端口 8800 并开始监听
ServerSocket serverSocket=new ServerSocket(8800);
// 使用 accept() 方法等待客户端发起通信
Socket socket=serverSocket.accept();
// 打开输入 / 输出流
InputStream is=socket.getInputStream();
OutputStream os=socket.getOutputStream();
// 获取客户端信息，即从输入流读取信息
BufferedReader br=new BufferedReader(new InputStreamReader(is));
String info=null;
while(!((info=br.readLine())==null)){
    System.out.println(" 我是服务器，客户登录信息为："+info);
}
// 给客户端一个响应，即向输出流中写入信息
String reply=" 欢迎你，登录成功 !";
os.write(reply.getBytes());
// 关闭资源
……
```

输出结果如图 5.15 所示。

图 5.15　用户登录

在示例 1 中，传递的数据均采用字符串的形式，而 Java 语言是面向对象的，如何在 Socket 中实现对象的传递呢？下面的示例 2 将升级示例 1，把字符串的数据修改为对象来传递。注意关键代码部分标粗的位置是表示代码修改的地方。

示例 2

升级示例 1，实现传递对象信息。

实体类关键代码：

```
import java.io.Serializable;
```

```
public class User implements Serializable{
    private String loginName;     // 用户名
    private String pwd;           // 用户密码
    public User() {
    }
    public User(String loginName, String pwd) {
       super();
       this.loginName=loginName;
       this.pwd=pwd;
    }
    // 省略 getter/setter 方法
    ……
}
```

客户端关键代码：

```
// 建立客户端 Socket 连接，指定服务器的位置以及端口
Socket socket=new Socket("localhost",8800);
// 打开输入 / 输出流
OutputStream os=socket.getOutputStream();
InputStream is=socket.getInputStream();
// 对象序列化
ObjectOutputStream oos=new ObjectOutputStream(os);
// 发送客户端登录信息，即向输出流中写入信息
User user=new User();
user.setLoginName("Tom");
user.setPwd("123456");
oos.writeObject(user);
socket.shutdownOutput();
// 接收服务器端的响应，即从输入流中读取信息
……
// 关闭资源
……
```

服务器端关键代码：

```
// 建立一个服务器 Socket（ServerSocket），指定端口并开始监听
ServerSocket serverSocket=new ServerSocket(8800);
// 使用 accept() 方法等待客户端发起通信
Socket socket=serverSocket.accept();
// 打开输入 / 输出流
InputStream is=socket.getInputStream();
OutputStream os=socket.getOutputStream();
// 反序列化
ObjectInputStream ois=new ObjectInputStream(is);
// 获取客户端信息，即从输入流中读取信息
User user=(User)ois.readObject();
if(!(user==null)){
```

```
        System.out.println(" 我是服务器，客户登录信息为："+user.getLoginName()+", "+
        user.getPwd());
    }
    // 给客户端一个响应，即向输出流中写入信息
    ……
    // 关闭资源
    ……
```

示例 1 和示例 2 基本完成了客户端和服务器端的交互，采用一问一答的模式，先启动服务器进入监听状态，等待客户端的连接请求，连接成功以后，客户端先“发言”，服务器给予“回应”。

2. 实现多客户端用户登录

这样一问一答的模式在现实中显然不是人们想要的。一个服务器端不可能只针对一个客户端服务，一般是面向很多的客户端同时提供服务，但是单线程实现必然是这样的结果。解决这个问题的办法是采用多线程的方式，可以在服务器端创建一个专门负责监听的应用主服务程序、一个专门负责响应的线程程序。这样就可以利用多线程处理多个请求。

下面仍以用户登录为例来介绍如何处理多客户端的请求。

示例 3

升级示例 2，实现多客户端的响应处理。

分析如下。

（1）创建服务器端线程类，run() 方法中实现对一个请求的响应处理。

（2）修改服务器端代码，让服务器端 Socket 一直处于监听状态。

（3）服务器端每监听到一个请求，创建一个线程对象并启动。

线程类关键代码：

```
public class LoginThread extends Thread {
   Socket socket=null;
   // 每启动一个线程，连接对应的 Socket
   public LoginThread(Socket socket){
      this.socket=socket;
   }
   // 启动线程，即响应客户请求
   public void run(){
      try {
         // 打开输入 / 输出流
         // 反序列化
         // 获取客户端信息，即从输入流中读取信息
         // 给客户端一个响应，即向输出流中写入信息
         // 关闭资源
         ……
      } ……
   }
}
```

服务器端关键代码：

```
// 建立一个服务器 Socket（ServerSocket）指定端口并开始监听
ServerSocket serverSocket=new ServerSocket(8800);
// 使用 accept() 方法等待客户端发起通信
Socket socket=null;
// 监听一直进行中
while(true){
   socket=serverSocket.accept();
   LoginThread LoginThread=new LoginThread(socket);
   LoginThread.start();
}
```

创建多个客户端程序，启动运行。

输出结果如图 5.16 所示。

图 5.16　多客户端用户登录

至此，任务 2 已经基本可以完成了。接下来，再来学习另外一个重要的类。

3．InetAddress 类

java.net 包中的 InetAddress 类用于封装 IP 地址和 DNS。要创建 InetAddress 类的实例，可以使用工厂方法，因为此类没有可用的构造方法。表 5-3 中列出了常用的工厂方法。

表5-3　InetAddress类中的工厂方法

方　法	说　明
static InetAddress getLocalHost()	返回表示本地主机的InetAddress对象
static InetAddress getByName(String hostName)	返回指定主机名为hostName的InetAddress对象
static InetAddress[] getAllByName(String hostName)	返回主机名为hostName的所有可能的InetAddress对象组

如果找不到主机，两种方法都将抛出 UnknownHostNameException 异常。

下面通过示例 4 学习 InetAddress 类的用法。

示例 4

输出本地主机的地址信息。

关键代码：

```
class InetAddressTest {
   public static void main(String args[]) {
      try {
       InetAddress add=InetAddress.getLocalHost();
       System.out.println(" 本地主机的地址是："+add);
      }
      catch (UnknownHostException u) {}
   }
}
```

任务 3　实现基于 UDP 协议的 Socket 编程

关键步骤如下。

- 利用 DatagramPacket 对象封装数据包。
- 利用 DatagramSocket 对象发送数据包。
- 利用 DatagramSocket 对象接收数据包。
- 利用 DatagramPacket 对象处理数据包。

5.3.1　基于 UDP 协议的 Socket 编程

基于 TCP 的网络通信是安全的，是双向的，如同拨打 10086 服务电话，需要先有服务端，建立双向连接后，才开始数据通信，而 UDP 的网络通信就不一样了，只需要指明对方地址，然后将数据送出去，并不会事先连接。这样的网络通信是不安全的，所以只应用在如聊天系统、咨询系统等场合下。

在开始学习基于 UDP 协议的网络通信之前，先了解术语“数据报”。数据报是表示通信的一种报文类型，使用数据报进行通信时无须事先建立连接，它是基于 UDP 协议进行的。

Java 中有两个可使用数据报实现通信的类，即 DatagramPacket 和 DatagramSocket。DatagramPacket 类起到数据容器的作用，DatagramSocket 类用于发送或接收 DatagramPacket。

DatagramPacket 类不提供发送或接收数据的方法，而 DatagramSocket 类提供 send() 方法和 receive() 方法，用于通过套接字发送和接收数据报。下面分别介绍两个类的构造方法及一些常用方法。

1．DatagramPacket 类

（1）构造方法

客户端要向外发送数据，必须首先创建一个 DatagramPacket 对象，再使用 DatagramSocket 对象发送。DatagramPacket 类的常用构造方法如表 5-4 所示。

表5-4　DatagramPacket类的常用构造方法

构造方法	说　明
DatagramPacket(byte[] data, int size)	构造DatagramPacket对象，封装长度为size的数据包
DatagramPacket(byte[] buf, int length, InetAddress address, int port)	构造DatagramPacket对象，封装长度为length的数据包并发送到指定的主机、端口号

（2）常用方法

DatagramPacket 类的常用方法如表 5-5 所示。

表5-5　DatagramPacket类的常用方法

方　法	说　明
byte[] getData()	返回字节数组，该数组包含接收到或要发送的数据报中的数据
int getLength()	返回发送或接收到的数据的长度
InetAddress getAddress()	返回发送或接收此数据报的主机的IP地址
int getPort()	返回发送或接收此数据报的主机的端口号

2. DatagramSocket 类

（1）构造方法

DatagramSocket 类不维护连接状态，不产生输入 / 输出数据流，它的唯一作用就是接收和发送 DatagramPacket 对象封装好的数据报。常用的 DatagramSocket 类的构造方法如表 5-6 所示。

表5-6　DatagramSocket类的常用构造方法

构造方法	说　明
DatagramSocket()	创建一个DatagramSocket对象，并将其与本地主机上任何可用的端口绑定
DatagramSocket(int port)	创建一个DatagramSocket对象，并将其与本地主机上指定的端口绑定

（2）常用方法

DatagramSocket 类的常用方法如表 5-7 所示。

表5-7　DatagramSocket类的常用方法

方　法	说　明
void connect(InetAddress address, int port)	将当前DatagramSocket对象连接到远程地址的指定端口
void close()	关闭当前DatagramSocket对象

续表

方　法	说　明
void disconnect()	断开当前DatagramSocket对象的连接
int getLocalPort()	返回当前DatagramSocket对象绑定的本地主机的端口号
void send(DatagramPacket p)	发送指定的数据报
void receive(DatagramPacket p)	接收数据报。收到数据以后，存放在指定的DatagramPacket对象中

5.3.2　使用 Socket 编程实现客户咨询

利用 UDP 通信的两个端点是平等的，也就是说通信的两个程序关系是对等的，没有主次之分，甚至它们的代码都可以完全一样，这一点要与基于 TCP 协议的 Socket 编程区分开来。

基于 UDP 协议的 Socket 网络编程一般按照以下 4 个步骤进行。

（1）利用 DatagramPacket 对象封装数据包。

（2）利用 DatagramSocket 对象发送数据包。

（3）利用 DatagramSocket 对象接收数据包。

（4）利用 DatagramPacket 对象处理数据包。

接下来实现一个最简单的使用 UDP 通信的程序。

示例 5

模拟客户咨询功能，实现发送方发送咨询问题，接收方接收并显示发送来的咨询问题。

分析如下。

实现这个功能，首先要充分理解这句话：基于 UDP 通信的两个程序关系是对等的，没有主次之分；其次理解 DatagramPacket 和 DatagramSocket 这两个类，DatagramPacket 类起到数据容器的作用，DatagramSocket 类用于发送或接收 DatagramPacket。

发送方实现步骤如下。

1）获取本地主机的 InetAddress 对象。

2）创建 DatagramPacket 对象，封装要发送的信息。

3）利用 DatagramSocket 对象将 DatagramPacket 对象数据发送出去。

发送方关键代码：

```
InetAddress ia=null;
String mess=" 你好，我想咨询一个问题。";
// 获取本地主机地址
ia=InetAddress.getByName("localhost");
// 创建 DatagramPacket 对象，封装数据
DatagramPacket dp=new DatagramPacket(mess.getBytes(),mess.getBytes().length, ia,8800);
// 创建 DatagramSocket 对象，向服务器发送数据
```

```
DatagramSocket ds=new DatagramSocket();
ds.send(dp);
// 关闭 DatagramSocket 对象
ds.close();
```

接收方实现步骤如下。

1）创建 DatagramPacket 对象，准备接收封装的数据。

2）创建 DatagramSocket 对象，接收数据保存于 DatagramPacket 对象中。

3）利用 DatagramPacket 对象处理数据。

接收方关键代码：

```
// 创建 DatagramPacket 对象，用来准备接收数据包
byte[] buf=new byte[1024];
DatagramPacket dp=new DatagramPacket(buf,1024);
// 创建 DatagramSocket 对象，接收数据
DatagramSocket ds=new DatagramSocket(8800);
ds.receive(dp);
// 显示接收到的信息
String mess=new String(dp.getData(),0,dp.getLength());
System.out.println(dp.getAddress().getHostAddress()+" 说："+mess);
```

输出结果：

127.0.0.1 说：你好，我想咨询一个问题。

示例 5 简单实现了基于 UDP 的网络编程。实际上只要明确了原理和步骤，再复杂的需求只要分解也能轻松实现，再来看下面的这个示例。

示例 6

升级示例 5，发送方发送咨询问题，接收方回应咨询。

分析如下。

这个功能实际上就是有来有往的过程，只需要在接收方接收到问题后，给发送方一个回应，然后发送方接收回应即可。简单来说，接收方变成发送方，发送一个回应，发送方变成接收方，接收这个回应。将代码互相复制修改就可以了。这也是任务 3 的实现思路，运行效果如图 5.17 所示。

图 5.17　在线客服系统

任务 4　搭建 JUnit 测试框架

关键步骤如下。

- 测试工具的选择。
- 测试用例的选择。
- 测试用例的编写。

5.4.1　认识软件测试

在编写程序的过程中，代码完成以后必须进行测试和调试，也就是说程序员要对自己编写的代码负责，既要保证代码的正确编译运行，又要保证与预期结果相符合，这就涉及到单元测试，下面将介绍测试相关的内容。

1. *软件测试的意义*

什么是软件测试呢？测试是发现并指出软件（包括建模、需求分析、设计等阶段产生的各种文档产品）中存在的缺陷的过程。这个过程指出软件中缺陷的确定位置，进行详细记录，并且同时给出与预期的结果偏差。一般软件测试采用人工或利用工具来完成。测试在软件开发周期中起着至关重要的作用：

- 测试可以找到软件中存在的缺陷，避免连锁负面反应的发生。
- 测试不仅为了找到缺陷，更能帮助管理者提升项目管理能力并预防缺陷的发生，改进管理措施。
- 测试是评定软件质量的一个有效方法。

2. *软件测试的分类*

根据测试的角度不同，软件测试有不同的分类标准：

- 从是否关心软件内部结构和具体实现角度，可分为白盒测试和黑盒测试。
- 从软件开发过程的阶段，可分为单元测试、集成测试和确认测试等。

简单介绍这几种测试的偏重点。

（1）白盒测试

白盒测试也称为结构测试或逻辑驱动测试，它按照程序内部的结构来测试程序，这种方法是把程序看成一个打开的盒子，测试人员对程序内部的结构需要非常清晰明确。

（2）黑盒测试

黑盒测试也称为功能测试，它通过测试来检测每个功能是否能够正常使用。在测试中，把程序看成一个不能打开的盒子，测试人员在完全不考虑程序的内部结构和内部特性的基础上，进行功能上的测试。

（3）单元测试

测试人员需要依据详细设计说明书和源程序清单，了解该模块的需求、条件和逻辑结构，对软件中最小可测试单元进行检查和验证。所以，对于单元测试，很多都由程序

员自己来完成。

5.4.2 JUnit 测试框架

1. JUnit 测试框架概述

如今单元测试在面向对象的开发中变得越来越重要，而一个简单易用、功能强大的单元级测试框架的产生为广大程序员带来了全新的测试体验。在 Java 编程环境中，JUnit 测试框架（JUnit Framework）是一个已经被多数 Java 程序员采用和证实了的优秀测试框架。

JUnit 是由 Erich Gamma 和 Kent Beck 共同编写的一个回归测试框架，它的主要特性包括如下方面。

- 用于测试期望结果的断言。
- 用于共享测试数据的测试工具。
- 用于方便地组织和运行测试的测试套件。
- 图文并茂的测试运行器。

同时，JUnit 是在极限编程和重构中被极力推荐使用的工具，因为它的优势突出，主要体现在以下几个方面。

- 测试代码和产品代码分开。
- 易于集成到测试人员的构建过程中。
- 开放源代码，可以进行二次开发。
- 可以方便地对 JUnit 的功能进行扩展。

下面就来介绍这个强大的单元测试框架。

2. JUnit 测试框架包介绍

JUnit 由 SourceForge 发行，从其官方网站上可以查阅 JUnit 相关的帮助文档，同时可以下载 JUnit 安装 jar 包，目前很多开发 IDE 环境都集成了该 jar 包，开发时只需要导入即可。

JUnit 测试框架包中包含了 JUnit 测试类所需的所有基类，实际上这个包也是整个 JUnit 的基础框架。

下面就来介绍该包中最基本的几个类和接口。

（1）Assert 静态类

Assert 类包含了一组静态的测试方法，用来比对期望值和实际值逻辑，验证程序是否正确（这个过程称之为断言），若测试失败则标识未通过测试。Assert 类提供了 JUnit 使用的一整套断言，是一系列断言方法的集合。

（2）Test 接口

Test 接口使用了 Composite 设计模式，TestCase 类和 TestSuite 类都实现了此接口。所有实现 Test 接口的类都要实现 run() 方法，该方法执行测试，并使用 TestResult 实例接收测试结果。

（3）TestCase 抽象类

TestCase 抽象类代表一个测试用例，负责测试时对客户类的初始化以及测试方法的

调用。启动 TestCase 的 run() 方法后即创建了一个 TestResult 实例。它是 JUnit 测试框架的核心部分。

（4）TestSuite 测试包类

TestSuite 测试包类代表需要测试的一组测试用例，负责包装和运行所有的测试类。因为对每一个类的测试，可能是测试其多个方法，也可能是对多个方法的组合测试，TestSuite 测试包类就负责组装这些测试。

（5）TestRunner 运行包类

TestRunner 运行包类是运行测试代码的运行器。

（6）TestResult 结果类

TestResult 结果类集合了任意测试累加结果，测试时将 TestResult 实例传递给每个测试的 run() 方法来保存测试结果。

5.4.3　JUnit 3.x 测试框架

1. JUnit 3.x 测试框架概述

前面已经提到，测试对于保证软件开发质量有着非常重要的作用，而单元测试是必不可少的测试环节。JUnit 是一个非常强大的单元测试包，可以对一个或多个类的单个或多个方法进行测试，还可以将不同的 TestCase 组合成 TestSuite，是将测试任务自动化的工具。MyEclipse 中集成了 JUnit，可以非常方便地编写 TestCase。JUnit 3.x 中会自动执行 test 开头的方法，这是依赖反射执行的。

2. 使用 JUnit 3.x 进行单元测试

搭建 JUnit 3.x（.x 代表版本）测试框架，必须了解以下几个方法的作用。

- testXxx()：JUnit 3.x 自动调用并执行的方法，必须声明为 public 并且不能带参数，必须以 test 开头，返回值为 void。
- setUp()：初始化，准备测试环境。
- tearDown()：释放资源。

它们的调用顺序为 setUp() → testXxx() → tearDown()。

使用 JUnit 3.x 进行单元测试一般按照以下 3 个步骤进行。

（1）在 Java 工程中导入所需要的 JUnit 测试 jar 包，选中 setUp() 方法和 tearDown() 方法。

（2）在 Java 工程中选中要测试的方法并完成测试类的方法编写。

（3）执行程序，红色表示失败，绿色表示成功。

示例 7

使用 JUnit 3.x 测试一个计算两个数相加的方法。

如下为一个简单的 Calculate 类，类中有一个方法 add()，方法具体代码如下所示。

```
public int add(int a,int b){
   return a+b;
}
```

根据 JUnit 的要求，测试方法的定义必须如下所示。

```
public void  testAdd(){
}
```

关键代码：

```
import junit.framework.Assert;
import junit.framework.TestCase;
public class CalculateTest extends TestCase {
   private Calculate cal;
   protected void setUp() throws Exception {
      cal=new Calculate();
   }
   protected void tearDown() throws Exception {
      super.tearDown();
   }
   // 测试断言方法
   public void testAdd(){
      // 断言 add() 方法 1+2=3
      Assert.assertEquals(cal.add(1,2), 3);
   }
}
```

如上就是一个最简单但又最典型的使用 JUnit 3.x 进行单元测试的例子。JUnit 测试框架不但允许进行一个 TestCase 的测试，还允许将多个 TestCase 组合成 TestSuite，让整个测试自动完成。

5.4.4 JUnit 4.x 测试框架

1. JUnit 4.x 测试框架概述

在前面的学习中，大家已经对 Annotation 注解有了一个比较深刻的认识。接下来要介绍的 JUnit 4.x 就与 Annotation 有很大的联系。JUnit 4.x 对 JUnit 测试框架进行了颠覆性的改进，JUnit 4.x 主要利用 JDK 5.0 中的新特性 Annotation 的特点来简化测试用例的编写。

2. 使用 JUnit 4.x 进行单元测试

首先看一下使用 JUnit 4.x 如何实现示例 7。

注意

JUnit 4.x 搭建测试框架的步骤与 JUnit 3.x 是一样的，请大家参照搭建。

示例 8

使用 JUnit 4.x 完成测试一个计算两个数相加的方法。

关键代码：

```
public class CalculateTest {
```

```
    private Calculate cal;
    @Before
    protected void setUp() throws Exception {
        cal=new Calculate();
    }
    @After
    protected void tearDown() throws Exception {
        super.tearDown();
    }
    @Test
    public void Add(){
        Assert.assertEquals(cal.add(1,2), 3);
    }
}
```

注意

采用 Annotation 的 JUnit 4.x 不要求必须继承 TestCase，而且测试方法也不必以 test 开头，只要以 @Test 这样的元数据来描述即可。

在上面的示例 8 中，大家是否注意到了多以 @ 符号开头的各种元数据呢？这就是 JUnit 4.x 中的注解 Annotation，其实，JUnit 4.x 是一个全新的 JUnit 测试框架，而不是旧框架的升级版本。JUnit 4.x 中的注解作用，简单来说，就是起到指示测试程序测试步骤及标志的作用。

JUnit 4.x 中的常用注解如表 5-8 所示。

表5-8　JUnit 4.x的常用注解

注解标识	说　明
@Before	用于标注每一个测试方法执行前都要执行的方法
@After	用于标注每一个测试方法执行后都要执行的方法
@Test	用于标注一个测试方法
@Ignore	用于标注暂不参与测试的方法
@BeforeClass	标注的方法在整个类的所有测试方法运行之前运行一次
@AfterClass	标注的方法在整个类的所有测试方法运行结束之后运行一次

综上所述，可以对 JUnit 3.x 和 JUnit 4.x 进行单元测试的区别总结如下。

- JUnit 3.x 中的测试类必须继承 TestCase，而 JUnit 4.x 则不是必需的。
- JUnit 3.x 的测试类必须重写 TestCase 的 setUp() 方法和 tearDown() 方法，分别执行初始化和释放资源的操作。而相同的工作，在 JUnit 4.x 中是使用 @Before 和

@After 来标识的，并且方法名可以随意定义。

- JUnit 3.x 中对某个类的某个方法进行测试时，测试类对应的测试方法名是固定的，必须以 test 开头，而 JUnit 4.x 中则不是这样，只需要使用 @Test 标识即可。
- 使用 JUnit 4.x 进行测试用例的编写相对灵活，和编写一个普通类没有什么区别，只需要加上注解标注即可，这种松耦合的设计理念相当优秀。

5.4.5 测试套件

JUnit 测试框架提供了一种批量运行测试类的方法，称为测试套件。简单地说，测试套件就是把几个测试类打包组成一套数组进行测试。这样，每次需要验证系统功能正确性时，只执行一个或几个测试套件即可。测试套件的写法非常简单，只需要遵循以下规则即可：

（1）创建一个空类作为测试套件的入口，保证这个空类使用 public 修饰，而且存在公开的不带有任何参数的构造方法。

（2）使用注解 org.junit.runner.RunWith 和 org.junit.runners.Suite.SuiteClasses 修饰这个空类。

（3）将 org.junit.runners.Suite 作为参数传入注解 RunWith，以提示 JUnit 使用套件运行器执行此类。

（4）将需要放入此测试套件的测试类组成数组作为注解 SuiteClasses 的参数。

测试套件中不仅可以包含基本的测试类，还可以包含其他测试套件，这样可以很方便地分层管理不同模块的单元测试代码。

本章总结

本章学习了以下知识点。

- 网络中使用 IP 地址唯一标识一台计算机。IP 地址由网络部分和主机部分共同组成，常用的 IP 地址有 A、B、C 这 3 类。
- 网络编程作为 Java 中的主要应用之一，可以通过使用 java.net 包来实现。
- TCP/IP 套接字是最可靠的双向流协议。在等待客户端请求的服务器使用 ServerSocket 类，而要连接至服务器的客户端则使用 Socket 类。
- 基于 UDP 的网络编程中，DatagramPacket 是起到数据容器作用的一个类，DatagramSocket 是用于发送或接收 DatagramPacket 的类。
- InetAddress 是一个用于封装 IP 地址和 DNS 的类。
- 简单易用、功能强大的单元级测试框架—JUnit 测试框架（JUnit Framework）是一个已经被多数 Java 程序员采用和证实了的优秀测试框架。

本章练习

1．简述域名解析原理。

2．编写程序，查找指定域名为 www.taobao.com 的所有可能的 IP 地址。输出结果如图 5.18 所示。

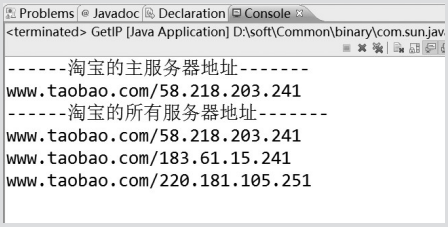


图 5.18　淘宝网的所有服务器 IP 地址

3．模拟用户登录，预设用户数据，提示登录成功或不成功的原因。输出结果如图 5.19 至图 5.22 所示。

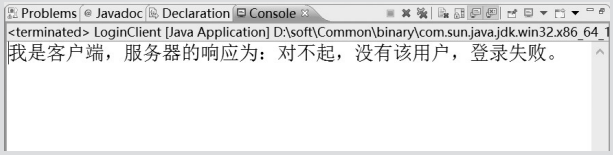


图 5.19　登录失败客户端显示

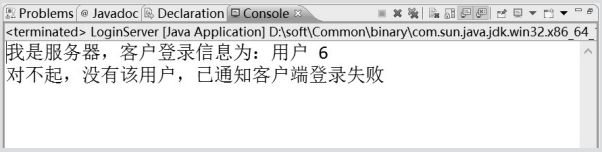


图 5.20　登录失败服务器端显示

图 5.21　登录成功客户端显示

图 5.22　登录成功服务器端显示

4．针对示例 8，若关键代码（粗体部分）修改为如下形式，请问编译是否会报错？

```
public class CalculateTest {
    private Calculate cal;
    @Before
    protected void init() throws Exception {
        cal=new Calculate();
    }
    @After
    protected void final() throws Exception {
        super.tearDown();
    }
    @Test
    public void TAdd(){
        Assert.assertEquals(cal.add(1,2), 3);
    }
}
```