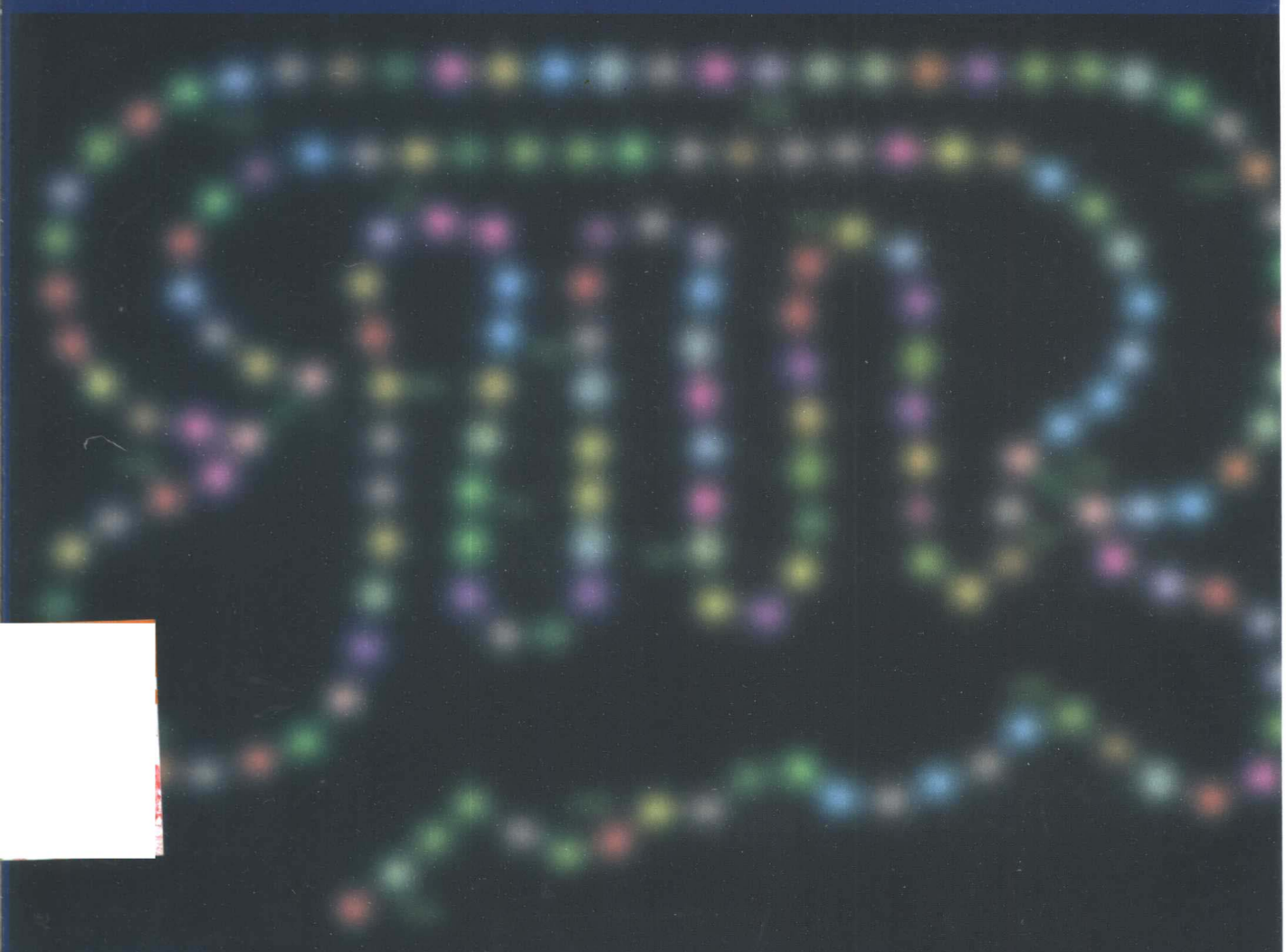
主编　殷兆麟

# Java网络编程

参编　姜淑娟　杨东平　秦国华　李洪涛



国防工业出版社

# Java网络编程

主编　殷兆麟

参编　姜淑娟　杨东平
　　秦国华　李洪涛

国防工业出版社

·北京·

**图书在版编目(CIP)数据**

Java网络编程/殷兆麟主编.—北京：国防工业出版社，2001.5

ISBN 7-118-02475-9

Ⅰ.J… Ⅱ.殷… Ⅲ.Java语言－程序设计 Ⅳ.TP312

中国版本图书馆CIP数据核字(2001)第04202号

国防工业出版社出版发行

(北京市海淀区紫竹院南路23号)

(邮政编码 100044)

北京奥隆印刷厂印刷

新华书店经售

*

开本 787 × 1092 1/16 印张 17 391千字

2001年5月第1版 2001年5月北京第1次印刷

印数：1—4000册 定价：24.00元



## 内 容 简 介

本书介绍了Java基本语法，Java面向对象程序设计技术，Java程序如何使用TCP和UDP通信协议实现一对一、一对多的网络通信，如何用URL类来获取网络上的文本、图像和相关的网络资源。在介绍这些基本知识基础上，介绍了Java程序如何通过JDBC访问网络数据库；Java远程方法调用机制（RMI）原理及其分布式应用；CORBA技术有关理论，远程接口描述语言IDL到Java的转换，以及如何利用Java在CORBA环境下实现分布式数据库应用。这些内容对于进一步学习网络系统集成技术及代理技术都是非常有用的。

本书可作为计算机专业本科高年级学生、研究生教材，也适用于计算机网络编程技术人员阅读。

# 前　言

计算机网络是一个空前活跃的领域，网络技术已广泛应用于各行各业。Java语言是一个广泛使用的网络编程语言，它具有可移植性、稳定性、安全性，并提供了并发机制、多线程，具有很高的技术性能。因此，Java语言目前已成为网络应用的主力开发语言，成为Internet上的“世界语”。为了适应广大计算机网络编程者的需要，本书介绍了Java网络编程技术。本书主要面向广大计算机专业的本科高年级学生、研究生，同时也适用于对计算机网络编程感兴趣的广大技术人员。

全书共分6章。第1章介绍Java语言特性及基本语法，Java中的面向对象程序设计技术；第2章主要介绍如何使用TCP和UDP通信协议实现一对一、一对多的网络通信；第3章则通过实例介绍如何用URL类来获取网络上的文本、图像、HTML文件和相关的网络资源。前3章目的是系统阐述Java网络编程有关的基本知识，是学习Java网络编程的基础。第4章介绍Java如何通过JDBC访问网络数据库；第5章介绍Java远程方法调用机制（RMI）原理及其分布式应用；第6章介绍CORBA技术有关理论，远程接口描述语言IDL到Java的转换，以及如何利用Java在CORBA环境下实现分布式数据库应用。最后两章内容对于进一步学习网络系统集成技术及代理技术（特别是移动代理技术）是非常重要的。

由于篇幅有限，示例程序源码未全部包括在本书中，有兴趣的读者可到http://www.cdcc.cumt.edu.cn/下载，或发E-mail至:zhlyin@cumt.edu.cn.询问。

由于成书时间仓促，作者水平有限，书中难免有不妥和错误之处，恳请广大读者指正。

殷兆麟　教授

于中国矿业大学计算机科学与技术系

# 目　录

# 第1章 Java 语 言

## 1.1 Java发展简史

Java语言（简称Java）是由Sun公司的James Gosling发明的，当时他是一个开发消费者电子工业工程的小组成员之一。为了实用，这种语言要产生简短、高效的执行代码。用它编写的程序必须容易地运行于不同类型和处理器上，为了减少使用这种语言开发和维护程序的成本，该语言必须是简单且面向对象的。

Java的突破是由Internet产生的，早期对可移植性的强调在于要求开发在Internet上下载的程序能够运行。因为这些程序可以不加修改地运行在任何能够解释Java语言的计算机上。用Java进行开发，意味着一个Web站点的设计者不必再编制同一程序的多个版本。

为了证明Java在软件开发上的价值，Sun公司使用Java开发了HotJava浏览器。尽管HotJava的最早版本非常粗糙且速度缓慢，但它说明了一个重要事实，即开发真正跨平台的程序是可能的。Java就是完成它的语言工具。然后Netscape把Java集成到了它的浏览器中，Java就像一枚火箭一样起飞了。

从一开始，Java的发展就得到来自于许多硬件和软件计算机的推动。比如，Netscape、IBM和Borland在许多领域对Java语言都做出了巨大的贡献。由于Sun公司对Java抱有无限的信心，使得它把Java的详细说明甚至源代码都公布于众，这样也就创造了第一个真正的标准的跨平台语言。

## 1.2 Java的语言特性

Java是一种跨平台的、适合于分布式计算机环境的面向对象的编程语言。它具有可移植、安全、面向对象、动态、高性能、简单、与体系结构无关性、动态执行等特性。

### 1.2.1 面向对象

面向对象是Java最重要的特性。Java语言的设计是完全面向对象的，它不是支持类似C语言那样的面向过程的程序设计技术。Java支持静态和动态风格的代码继承重用。单从面向对象的特性来看，Java类似于SmallTalk，但它的其他特性，尤其是适用分布式计算环境的特性却远远超越了SmallTalk。

“面向对象”目前是一个非常流行的术语，其影响领域从操作系统、编程语言及其开发环境、数据库管理系统直至软件工程。可以说，不管适合不适合，也不管是否真正具有面向对象特性，人们已经习惯了给自己的产品贴上“面向对象”的标签。但事实上，这些

产品有许多并不是真正面向对象的，或者说不是完全面向对象的。

何谓“面向对象”？面向对象其实就是现实世界模型的一个自然延伸。现实世界中的任何实体，都可以看作是对象。对象之间通过消息相互作用。传统过程式编程语言支持一个公式：

程序 = 算法 + 数据

面向对象编程语言也有一个公式：

程序 = 对象 + 消息

所有面向对象的编程语言支持3个概念：封装、多态性和继承。现实世界中的对象均具有属性和行为，映射到计算机程序，属性表示为数据，行为表示为程序代码。所谓封装，就是用一个自主式框架把代码和数据联编在一起，形成一个对象。也就是说，对象是支持封装的手段，是封装的基本单位。对象内的数据和代码，可以是共有的。私有代码和数据只能被对象其他部分访问，公有代码和数据则可以被其他对象访问。一般情况下，对象的公用部分是对象之间交互的机制。

多态性是指“一个对外接口，多个内在实现形式”的表示。多态性的一个典型例子是计算机中的堆栈。堆栈可以用来存储各种格式的数据，包括整数、浮点数和字符。不管存储的是何种数据，堆栈的算法实现却是一样的。在过程式编程中，必须针对每种数据类型专门实现一套堆栈表示和算法，名称和外部接口均必须不同。而在面向对象编程中，只需用一个对外接口和实现即可。针对不同数据类型，编程人员不必手工选择，只需使用统一接口名，系统可自动选择。

继承是指一个对象直接使用另一对象的所有属性和方法的过程。理解继承的一个很好的模型是现实世界中的家族树。事实上，很多实体都可以自上而下进行管理。例如，把个人计算机(PC)看作是一个实体，可以分为多个子实体，如Macintosh,IBM PC,Motorola 68000系列，等等。这些子实体均具有PC机的特性，PC机是这些子实体的父对象，这些子实体为PC机的子对象。继承可简化对象的定义，利用继承，在定义一个对象时，就可以只定义该对象独特于其父对象的特殊品质。

面向对象可以说是一种编程方法学。从这个意义上说，利用传统的面向对象编程语言，同样可以编写出具有面向对象的程序。经常使用的一些所谓面向对象编程语言，包括C++,Object Pascal(Borland Dephi中所用的编程语言)、Perl 5.0和Object C等，实际上只是一种混合型语言，是过程式语言加面向对象的扩展。这些语言支持对象的使用，但同样也支持过程式编程，事实上你几乎总是有意无意地使用一些过程式编程特征。

Java则不然，它是一门“纯”面向对象的编程语言。在一门“纯”面向对象的语言中，语言的任何方面都是基于消息或基于对象的；所有数据类型，无论简单还是复杂，均为对象类。Java实现了标准C语言中的所有数据类型，如整型、字符型和浮点型，这些基本数据类型可以作为对象，也可以不作为对象处理，除此之外的其他任何内容均为对象。这一方面保证了Java的高性能，另一方面又使得Java仍然是一门纯面向对象的编程语言。

### 1.2.2 可移植性

程序的可移植性指的是程序不经修改而在不同硬件或软件平台上运行的特性。在目前这种标准和开放系统备受推崇的时代，可移植性在一定程度上决定了程序的可应用性。反

过来说，标准的制定以及所谓开放系统的推出，其目的之一也是为了增强程序的可移植性。可移植性包括两种层次：源代码级可移植性和二进制级可移植性。C语言（本书简称为C）和C++语言（本书简称为C++）只具有一定程度的源代码级可移植性，这表明C或C++源程序要能够在不同平台（如DOS和Unix平台）运行，必须重新编译。

Java采用了多种机制来保证可移植性，其中最主要的有两条：

1）Java既是编译型又是解释型的。因此，Java编程人员在进行软件开发时，不必考虑软件运行平台。不仅开发的源代码是可移植的，甚至源代码经过编译之后形成的二进制代码——字节码，也同样是可移植的。任何一台机器只要配备了Java解释器，就可以运行Java二进制代码，而不管这种字节码是在何种平台上生成的。

2）Java采用的是基于国际标准的数据类型。Java的数据类型在任何机器上都是一致的，它不支持特定于具体的硬件环境的数据类型。Java的数据类型采用的是IEEE标准。比方说，Java的浮点数遵循的是IEEE 754标准。

如上所述，Java程序的最终执行需经过两个步骤：编译和解释。Java编译器所生成的可执行代码并不基于任何具体的硬件平台，而是基于一种抽象的机器——Java虚拟机（Java Virtual Machine——JVM）。JVM实际上是一个可运行Java代码的虚拟操作平台。通过预先把Java源程序编译成字节码，Java避免了传统的解释型语言的性能瓶颈，并确保了其可移植性。

除给出基于JVM的虚拟机器码外，Java语言还规定同一种数据类型在所有各种实现中必须占据相同的空间大小。C++的数据类型在不同的硬件环境或操作系统下占据的空间是不同的，如整数类型在Windows 3.11下为16位，而在Windows 95下却是32位。通过在数据类型的空间大小方面采用统一标准，Java成功地保证了其程序的平台独立性，或者所谓的“体系结构中立性”（Architecture Neutral）。

此外，Java的可移植性还体现在Java的运行环境上。Java编译器是用Java语言本身所编写的，而其他运行时环境则是用ANSI C编写的，整个运行时环境体现了一个定义良好的可移植性接口。最后，Java语言规范遵循POSIX标准，这也是使Java具有良好可移植性的重要原因。

Java的可移植性具有深远意义。长期以来，无论是对网络软件还是单机软件而言，使应用软件能够在任意平台上运行一直都是编程人员所梦寐以求的事情，如今Java终于使大家的梦想成真。有了Java开发工具，今后的编程人员在开发应用软件时，再不用分别开发IBM PC机版本、Macintosh版本和工作站版本等等，而只需开发一个“通用”的最终软件，这将大大加快软件产品的开发。

更进一步地讲，Java的可移植性还迎合了目前很时兴的“网络计算机”的思想。所谓“网络计算机”，就是指未来的专门用来连接到网络环境的计算机。这种机器使用廉价的芯片，没有硬盘或硬盘空间非常有限，依靠从网络服务器下载应用程序和内容来运行，运行结果也存放在服务器上。可以想象，如果各种应用软件都能用Java重新编写，并且放到某个Internet服务器中，那么未来的网络计算机的用户（同时也是Internet用户）将不再需要安装那些需占用大量空间的软件。他们只需有一个Java解释器，每当需要使用某种应用软件时，从Internet下载该软件的字节码即可，运行结果也可以发回到服务器。这在各种应用软件所需空间越来越大的今天无疑具有非凡的意义。

此外，这同时也给未来的软件产业带来了一种新的思路。今后的软件销售也许将不仅仅是今天这样的一次性销售的方式，到时候将可能出现一种“计时收费”的软件销售方式，用户在需要时才下载某个软件的字节码，并且只需要为自己的使用时间付费。这样，诸如版本更新、软件费用昂贵等问题，都将迎刃而解。

总之，Java的可移植性决定了这种新型语言将成为未来的网络环境上的“世界语”。

### 1.2.3 稳定性和安全性

分布式计算环境要求软件具有高度的稳定性和安全性。熟练的C++程序员，可能已经知道：C++程序在稳定性方面的最大问题，在于其指针的使用和缺乏主动的内存管理。这意味着：C++程序员完全可以编写出在语法和语义上均正确，但却能对系统产生巨大破坏作用的软件。

鉴于这些原因，为保证稳定性，Java采用了3个措施，首先，Java不支持指针数据类型，这样，程序员便不再能够凭借指针在任意内存空间，甚至是操作系统的内存空间中“遨游”；其次，它提供了数据下标的检查机制，从而使网络“黑客”们无法构造出类似C和C++语言所支持的那种指针；最后，Java还提供了自动内存管理机制，即一个自动的“内存垃圾”搜集程序。

Java运行时，环境保证字节码的传输过程中使用公开密钥加密机制(PKC)外，还提供了如下四级安全保障机制：

（1）字节码校验器（ByteCode Verifier）；

（2）类装载器（Class Loader）；

（3）运行时内存布局；

（4）文件访问限制。

当Java字节码进入解释器时，首先必须经过字节码校验器的检查，这是非常重要的。即使Java编译器生成的是完全正确的字节码，解释器也必须再次对其进行检查，因为在从Java程序的编译直到解释执行这段时间内，字节码可能被有意无意地改动过。

然后，Java解释器将决定程序中类的内存布局，这意味着“黑客”们将无法预先得知一个类的内存布局结构，从而也就无法利用该信息来“刺探”或破坏系统。随后，类装载器负责把来自网络的类封装到其单独的内存区域，避免应用程序之间的相互干扰或破坏作用。

最后，客户机一端管理员还可以限制从网络上装载的类只能访问某些被允许的文件系统。上述机制综合在一起，使得Java成了最安全的编程语言和环境之一，并且保证了Java代码无法成为类似特洛伊木马、病毒和蠕虫等具有潜在破坏作用的东西。

### 1.2.4 简单性

Java语言简单而有效，这与Java的起源有很大关系。Java最初是为对家用电器进行集成控制而设计的一种语言，因此它必须具有简单明了的特性。在一开始，Gosling等人曾经考虑过直接用C++编程语言，但由于C++过于复杂，存在大量冗余，再加上诸如安全性等

各种因素，才不得不忍痛割爱，并决定重新设计一种新的语言，也就是现在的Java。

Java的主要设计目标之一就是尽可能类似于C++，尽可能采用面向对象的编程风格。另一种设计目标则是希望从C++中消除其可能给软件开发、实现和维护带来麻烦的地方，包括其二义性、冗余和存在安全隐患之处，如操作符重载、多继承和数据类型自动转换等。综合来说，Java语言的简单性主要出于如下几种因素：

（1）Java的风格类似于C++，因而对C++程序员而言是非常熟悉的；从某种意义上来说，Java语言本身是C及C++的一个变种，因此，C++程序员可以很快掌握Java编程技术；

（2）Java摒弃了C++种容易引发程序错误的地方，如指针和内存管理；

（3）Java提供了自动内存垃圾搜集机制，从而减轻了编程人员的进行内存管理的负担，有助于减少软件错误；

（4）Java是完全面向对象的，它是最容易学习的面向对象编程语言之一；同时它还提供了大量可重用的类库。

Java的简单性是以增加运行时系统的复杂性为代价的。以内存管理为例，自动内存垃圾处理减轻了面向对象编程的负担，但Java运行时系统却必须内嵌一个内存管理模块。但无论如何，对编程人员而言，Java的简单性只会是一个优点，它可以使我们的学习曲线更趋合理化，加快我们的开发进度，减少程序出错的可能性。

### 1.2.5 高性能

正常情况下，可移植性、稳定性和安全性几乎总是以牺牲性能为代价的，解释型语言的执行效率一般也要低于直接执行源码的速度。但Java所采用的措施却很好地弥补了这些性能差距。这些措施如下。

1）多线程

线程也被称作“轻量(lightweight)进程”，线程的概念提高了程序执行的并发度，从而可提高系统效率。C和C++采用的是单线程的体系结构，均未提供对线程的语言级支持。与此相反，Java却提供了完全意义的多线程支持。

Java的多线程支持体现在两个方面。首先，Java环境本身就是多线程的，它可以利用系统的空闲时间来执行诸如必要的垃圾清除和一般性的系统维护等操作；其次，Java还提供了对多线程的语言支持。利用其多线程编程接口，编程人员可以很方便地写出支持多线程的应用程序，提高程序的执行效率。

必须注意的是，从Java语言规范可以看出，Java的多线程支持在一定程度上可能会受其运行时支持平台的限制，并且依赖于其他一些与平台相关的特性。比方说，如果操作系统本身不支持多线程，Java多线程就可能只是“受限”的或不完全的多线程。

2）高效的字节码

Java编译器生成的字节码和机器码的执行效率相差无几。Java字节码格式的设计充分考虑了性能因素，其字节码的格式非常简单，这使得经由Java解释器解释执行时可产生高效的机器码，据统计，Java字节码的执行效率非常接近于由C和C++生成的机器码的执行效率。

最后，Java运行时环境还提供了另外两种可选的性能提高措施：及时编译和嵌入C代码。及时编译是指在运行时把字节码编译成机器码，这意味着代码仍然是可移植的，但在一开始会有一个编译字节码的延迟过程。嵌入C代码在运行速度方面效果当然是最理想的，但会给编程人员带来额外负担，同时将降低代码的可移植性。不过不管怎么说，利用这些性能提高措施，Java的确可以达到与C++几乎相同的执行效率。

### 1.2.6 动态特性

Java的动态特性是其面向对象设计的延伸。这种特性使得Java程序能够适应不断变化的执行环境。类库的使用是最明显的一个例子。如果你曾经编写过C++程序，那么对类库的概念应该不会陌生。用C++编写的应用程序经常会用到各种类库，不仅仅是基础类库。很多时候还需要一些从第三方厂商处购买的类库。销售应用程序时，类库有时是单独出售的。对于C++应用程序，这就会导致一个问题：类库一旦升级，用这些类库编写的应用程序就必须重新编译，并且重新发送到用户手中，否则就无法利用升级后类库的新增功能。

Java的“滞后联编”却避免了这个问题。“滞后联编”机制使得Java完全利用了面向对象编程模式的优点。如前所述，Java程序的基本组成单元为类，这些类是在运行过程中动态装载的。因此，Java可以在分布式环境中动态地维护应用程序及其支持类库之间的一致性。这样，对于Java而言，其支持类库升级之后，相应的应用程序不必重新编译，也一样可以利用升级后类库的新增功能。

除此之外，Java的动态性还体现在其对动态数据类型和动态协议的支持上。利用一种特殊的Applet，即内容句柄，编程人员可很方便地使HotJava支持新的数据类型。类似地，通过编写协议句柄，可以使HotJava支持新的、自定义的传输协议。

Java的动态性有何价值？简而言之，Java的动态性使你能够真正拥有“即插即用”（plug-and-play）的软件模块功能。

### 1.2.7 分布式

分布式包括数据分布和操作分布。数据分布是指数据可以分散存放于网络上的不同主机，操作分布则指把计算分散由不同主机进行处理。Java支持WWW客户机／服务器计算模式。因此，它可以支持所有这两种分布性。

对于数据分布，Java提供了一个URL对象，利用此对象可以打开并访问网络上的对象，其访问方式与访问本地文件系统几乎完全相同。对于操作分布，Java客户机／服务器模式可以把运算从服务器分散到客户一端，提高整个系统的执行效率，避免瓶颈制约，增加动态可扩充性。

对于编程人员来说，Java的网络类库是对分布式编程的最好支持。Java网络类库是支持TCP/IP协议的子例程库。目前支持的协议有HTTP和FTP等。同时，通过编写协议句柄，编程人员还可以扩充Java支持的协议集合。

以上介绍了Java的主要特征。由这些特征可以看出，Java的确是一门适合Internet和分布式环境的编程语言。

## 1.3 Java语言的特点

### 1.3.1 包、接口和类

1）包

包是一组类和接口，它是用来管理庞大的命名空间和避免冲突的工具。每个类名和接口名都包含在某个包中。包名由分离的字段组成，段与段之间用“.”隔开。

Java语言提供一种机制，它可以使类和接口的定义和实现跨包有效。import关键字用于标志那些被当前包所引用的类。编译单元可以自动地引用本包中的每个类和接口。

使用import语句引用类、接口或者是包含它们的包的方法如下：

```
import Java.net.*;
```

它表示每个Java.net中的公共类都被引用。

2）接口

接口是方法定义和常量值的集合。Java通过接口可使处于不同层次的类具有相同行为。由于接口中只进行方法的说明，而不提供方法的实现，因此接口中所说明的方法，尚未实现方法体。接口可以说是抽象类的一种极端形式。接口可以提供对方法协议的封装，这些方法的实现没有限制在同一继承树上。当一个类实现接口时，它通常要实现接口中所说明的所有方法体（如果实现接口的类是抽象的——从来不实现——它将给它的子类留下部分或全部接口方法）。接口的一般定义格式如下：

```
interface InterfaceName [extends surperInterfaceList]          //接口说明
{
    type constantName=Value;            //常量说明，可有多个
    return Type methodName([paramList]); //方法说明，可有多个
}
```

接口能够解决一些多重继承性能处理的问题，相对比较起来在运行时开销较小。然而，由于接口涉及动态方法连接，在使用时必然有一些小的性能损失。

在使用接口时，允许多个类共享一个程序接口，使它们具有相同的行为，而无须考虑这些类之间的继承关系。

一个类也可以实现一到多个接口。当需要在类中实现接口时，必须借助implements关键字。如下所示：

```
public class ClassName extends superClass implements interface[,option interface]
{
  //class body
}
```

接口的概念给Java带来了许多好处：通过创建一个接口，可以先定义一个抽象类的协议，而不管其具体实现，即把具体实现放到以后再行定义，这在有些场合是非常有用的；其次，有了接口的概念，多个类可以共享相同的接口，相互之间不用管其他类是如何定义

该接口的方法。如果继承了接口的属性，其他用户就可以了解到如何调用这个类的方法。比如说，如果希望所有的Shapes类都有一个draw()方法，就可以创建一个接口，并把它命名为Shape。如下所示：

```
public interface Shape
{
    void draw();
}
public class Triangle implements Shape
{
    void draw()
    {
        //draw implementation
        }
    ...
}
```

这样，每当需要使用Shape时，就可以知道这个类有一个draw()方法，同时还可以确定这个方法都有哪些参数。

3）类

“类”(Class)是基于对象之上的一个概念，类本质上可以认为是对象的描述，是创建对象的“模板”。

类是组成Java程序的基本要素，类代表了经典的面向对象程序设计模型。在Java中，一个类的通常定义形式为：

```
[类的修饰字] class 类名 [extends 父类名]
{
.....
}
```

建立一个新类，程序员必须基于已有旧类的基础，新类是由已存在的类衍生出来的。衍生的类被称为子类，已有的类被称为其父类。类的衍生是可传递的。如果B是A的子类，C是B的子类，则C也是A的子类。

在类的说明中，用关键字extends来指明一个类的直接父类。

所有的类都是由单一的根类——Object衍生出来的，除Object类外，每个类只有一个直接父类。如果在说明一个类时没有指明它的直接父类，那么，就认为Object类是它的直接父类。

Java语言只支持单一继承性。但通过接口性能，它可以支持某些在其他语言中通过多继承性支持的性能。

### 1.3.2 出错与异常处理

对计算机程序来说，错误和异常情况都是不可避免的。因此，一门编程语言如果不提