国防科技大学出版社

JAVA
语言程序设计及应用

● 廖开际　马　并　罗端红　编著

JAVA 语言程序设计及应用

廖开际　马　并　罗端红　编著

国防科技大学出版社
·长沙·

图书在版编目(CIP)数据

JAVA 语言程序设计及应用/廖开际，马并，罗端红.— 长沙：国防科技大学出版社，1997.12
ISBN 7 - 81024 - 446 - 9

Ⅰ.JAVA 语言程序设计及应用
Ⅱ.廖开际　马　并　罗端红
Ⅲ.计算机 - JAVA 语言 - 程序设计
Ⅳ.TP312

国防科技大学出版社出版发行
电话:(0731)4555681　邮政编码:410073
E-mail:gfkdcbs@public.cs.hn.cn
责任编辑：何　晋　　责任校对：文　慧
新华书店总店北京发行所经销
湖南大学印刷厂印装
*
开本:787 × 1092　1/16　印张:15　字数:247 千
1997 年 12 月第 1 版第 1 次印刷　印数:1 — 5 000
*
定价：18.00 元

内 容 简 介

由 Sun Microsystem 推出的革命性编程环境 Java 具有强大的动画、多媒体和交互式编程能力，是一种简单的、灵活的、面向对象的网络编程语言。

全书共 12 章，从面向对象的基本概念入手，深入浅出地全面介绍了 Java 的特点、基本概念、语法结构、类库、小应用程序和图形、图像处理、动画及多线程等应用。示例丰富，分析透彻，各章后附有思考练习题。

本书兼顾了基础和提高的有机结合，即使没学过 C/C++语言的读者，也可以毫无困难地学习本书；学过其他面向对象语言的读者，通过学习本书可以得心应手地使用 Java 编写各种高级应用程序。

本书可作为相关专业本科生及培训班和非计算机专业研究生的教材，也可供广大计算机爱好者和技术人员使用。

目　　录

第一章　面向对象程序设计与 Java 语言概述

1.1　面向对象程序设计的基本概念

在早期的编程环境中，当程序员写一个程序时，变量的数量以及操作随着程序规模的增大而迅速增大，使人们只能将程序当成一系列线性的步骤。这种面向过程的模型使人们更多地考虑问题中的每一具体步骤而忽略对抽象问题进行模型化。过程性语言如 C 和 Fortran 采用这种数学模型自 60 年代以来获得了巨大的成功。这种模型对简单问题是比较合适的，但随着程序规模的增大，问题也越来越多。接口定义的不足或模块间的交互使程序难于管理并且错误倍出。

为了减少同时出现的事物数量，程序员不得不采取更有效的办法管理他们的代码，一种新的方法应运而生：面向对象的程序设计。这种将复杂系统进行分层抽象的方法不仅适用于现实世界而且也适用于计算机程序。传统的算法程序可以抽象成各种要处理的对象，一系列处理步骤可构成独立的对象间的信息集。每个对象封装了自己的行为。我们将这些对象甚至抽象对象当作现实世界具体的实例，它们能响应外界的刺激并进行相应的动作。这就是面向对象编程的基础。

正像人类理解复杂事物的方式一样，面向对象的概念构成了 Java 的核心。一旦明白了如何将面向对象概念应用于编程之后，就可以开始用 Java 编程了 。

面向对象程序设计围绕几个主要概念：抽象数据类型、类、类的层次(子类)、继承性和多态性。类和继承性是符合人们一般思维方式的描述模式。

1.1.1　什么是对象(Object)?

面向对象程序设计，就是希望能将生活中自然的概念，应用到程序设计上。现实生活中，什么是一个对象呢？其实，对象是你眼前所看到的任何物体，如一部计算机、一部车、你手上的这本书等。任何一个对象都有两个基本的特点：一个是物体内部的状态，譬如说，汽车现在是“停止的”，计算机现在是“开着的”等等。而另外一个特点则是对象对于外界给予的信息或操作方式，能用一套自己的方法来处理。譬如说，转动汽车上的启动钥匙，汽车就会发动起来等。面向对象的方法就是给程序中现实的对象定出模型。

一个程序中的对象具有自己的内部数据，可以把这些数据想像成现实对象的状态，状态是不能随便改变的，必须由外界向对象传递信息，再由对象按照既定的方法加以改变后才会变成新的状态。同样的，对于一个程序对象，如果想要改变其内部的数据，也必须将外界的指令信息，传给这个程序对象原来定义的方法，由对象本身的方法来修正内部的数据，并给予外界适当的反应。图 1.1 给出了程序对象模型示意图。

对象通常作为计算机模拟思维，表示真实世界的抽象。一个对象像一个软件构成块，它包含了数据结构和提供相关的行为(操作)。对象本身可为用户提供一系列服务——可以改变对象状态、测试、传递消息等，用户无须知道服务的任何实现细节，操作完全是封闭的。

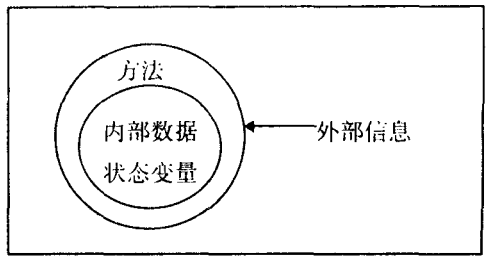

图 1.1　程序对象示意图

1.1.2　类、对象和封装

一个类描述涉及到定义该类任一对象的一个实例的所有行为特征。实际上，一个类定义的是一种对象类型。

可以把类(class)想象成是对象的基本原型，同一种类的对象有着同样的方法和性质，譬如说，你可以定义一个“汽车”的类，这个类中定义每一辆车子必须有颜色、有车门等，对于踩油门的反应都是车子的速度增快，那么，由这个“汽车”的类原型，就可以生产出许多汽车的对象实体(instance)。这些汽车的对象实体都有自己的颜色(虽然这些颜色可能都不一样)，也都有自己的车号，但是这些性质都存在，更不用说对于踩油门踏板的反应了。

抽象数据类型是面向对象程序设计的中心概念之一。一个抽象数据类型是一个模型，此模型包含一个类型和与之相关的操作集。定义这些操作集的是基本类型的行为。

在大多数面向对象语言中，类的定义描述以抽象数据类型为基础的对象行为，抽象数据类型定义了能以类型为基础执行所有操作的接口。类定义也指定了实现细节或类型的数据结构。通常这些实现细节仅在该类范围内存取，我们称这种类型为私有(private)类型。当数据类型的全部或部分在该类范围外存取，我们称这样的类型为公共(public)类型。

按面向对象的说法，为一个类而定义的所有操作称为方法(methods)。这些方法类似于非面向对象语言中的过程和函数。如果一个类的 Public 操作能在许多应用领域中被充分利用，那么可以说类是组成可重用软件程序块的基础。

一个对象(object)是被一个特定类说明的变量。这样一个对象通过包含在类定义中的所有域的拷贝(private 和 public 两部分)来封装。通过访问一个或几个在类定义上的方法，可以对这个对象实施各种操作。引用一个方法的过程称之为向这个对象发送一个消息(message)。一个典型的消息包含一些参数，正像在非面向对象程序语言中过程或函数调用(或引用)的一些参数一样，一个典型引用方法的操作是修改存储于特定对象中的数据。

每个类变量或对象表示该类的一个实例(instance)。如果几个对象被定义具有同样的类，它们将是包含有不同值的一个集合。

面向对象方法通过子类提供类型的等级层次。一个子类(subclass)定义一个对象集合的行为，该对象继承父类(parent class)的各种特征，子类反过来又将它自己的或继承来的特征传递到它的子类中。借助允许建立子类的方法，可使软件开发的周期缩短，从而

降低开发复杂性和费用。

子类能导致增加问题求解的能力。用基本类的集合建立子类代替修改现成的软件，或坏的软件，或重写的软件。这些新的子类的对象组成了软件构造的从属结构。

一个类定义的私有部分通常定义了以数据类型为基础的数据结构。仅仅在该类范围内部可以存取指定的方法，这些方法经常支持公共方法的实现。有时一个类的私有部分可以包含其它类的对象，这个其它类仅在给定类的范围内部才能被操纵。

一个类的公共部分通常指定对方法的接口，在许多交叉应用领域中它们构成了这个类的可重用性的基础。这些方法能引用到该类的范围之外，当然是用发送消息到给定类的对象的方式。

封装是用于被定义的各个软件对象的过程中。封装定义为:

1. 所有对象内部软件范围具有清晰的边界;
2. 描述该对象与其它对象如何相互作用的一个接口;
3. 受保护的内部实现。该实现给出了由软件对象提供的功能的细节，实现细节不能在定义该对象的类的范围外进行访问。

封装概念不仅涉及到类的描述，而且涉及到问题的求解过程。封装的单位是对象，该对象的特性由它自己的类说明来描述。这些特性为相同类的其它对象所共享，对象的封装比讲一个类表示的封装更具体化。有了封装这个定义，一个类的每一个实例(instance)在一个问题求解中是一个独立的封装，或称作组件(问题解的分量)。

Java 语言对类的定义方式和 C++的方式基本上是相同的，我们看下面的一小段例子即可得知:

程序 1-1

```
// Java 的类实例
class Car {
    int color-number ;      //车的颜色编号
    int number ;             //车号
    char owner ;             //车主
    int speed ;              //车速
    ...
    push-break ()            //踩煞车的反应
      { ... }
    Add-oil()                //加油的反应
      { ... }
}
```

1.1.3 子类——继承性和多态性

下面介绍类层次概念。在类层次体系中，某些类是从属于其它类，称为子类。子类是被认为层次体系组合下的类的具体情况，通常在类层次体系中的下层表示一个增加的详细说明，而高层表示更高的概括。

类的一个最大作用，就是它提供了“继承性”的实现方法。在面向对象程序设计中，所谓继承，是指被继承的类中，原定义的性质和方法都直接被后来的类所承接，同时后来的类还可以在原类中定义的性质和方法之外加入自己新的定义。就拿上一段中的“汽车”类来作说明，我们可以定义出一种新的类叫“垃圾车”，“垃圾车”类直接继承自“汽车”类，所以原来“汽车”类已定义过的性质和对外界信息的处理方法，在“垃圾车”类上同样有效。也就是“垃圾车”也有颜色，也有车号等，踩了油门一样会跑。但是垃圾车的特殊用途是装垃圾，则必须再独立定义。

可见，继承的最大好处是程序代码的复用。原先我们在“汽车”类里写过的定义，写过的程序，在“垃圾车”类中都不需要再次重复。另外一个好处则是可以帮助编程人员在写程序时，很自然地运用面向对象的概念去实现所要处理的数据，使程序和数据的组织更加完善。下面是定义“垃圾车”类的例子:

程序 1-2

```
//Java 的类继承实例
class Trash-Car extends Car
{
    double amount              //垃圾数量
    fill-trash()               //装垃圾的方法
      { … }
}
```

在大多数面向对象语言中，如果类 P 是子类 S 的一个父辈，则子类 S 的一个对象 s 能使用父类 P 的一个对象 p 。这蕴含着消息(即操作)公共集合能发送到类 P 和类 S 的各个对象。当同样的消息能被发送到父类的对象和它的子类的对象时，我们把此定义为多性态(polymorphism)。多态性允许每个对象响应公共消息格式，即用合适的方式从一个对象取来送到子类对象去。例如，一个打印方法可定义输出到赋予一个对象状态的某个数据域。多性态的相同信息 print ，被送到一个类和它的子类的所有对象，这样每个对象知道如何响应这个消息，也允许用其它子类对象的不同方式响应。例如，在不同数据域可以从每个不同的子类的对象输出。在不同类型的对象上，对一个类似的操作，使用相同消息的能力是与问题求解时人的思维方式相一致的。对打印整数、浮点数、字符、字符串和数据记录等，使用不同的术语是不自然的。多态性是对问题求解的面向对象方法中的关键特征之一。

方法重载是让类以统一的方式处理不同类型数据的一种手段。它是静态的，这是因为实现类并在编写方法之前要考虑到将要遇到的所有数据类型。在某种程度上，这是非常必要的而且也能导致清晰和可预料的代码，然而它不灵活，经常需要在许多代码被冻结和没有源代码的情况下扩充环境。

如果正确地应用了这种简明的面向对象概念，将可构造出比传统编程模型更强健更具扩充性的编程环境。定义良好的类层次是对经严格开发和测试过的代码进行重用的基础。数据和代码的封装使人们可以随时移植他们的工作而不会破坏基于公有接口的代码，多态性和接口允许人们创建简明的、合理的、可读的、一贯正确的代码。 Java 对

这些特性的实现有点像 simula 和 SmallTalk 的混合，而语法上则接近 C 和 C++ 。

1.2 Java 的发展史

Java 的产生最早可以追溯到 1991 年，当时 WWW 还没有正式发布。当初 Sun 公司希望能将公司从传统起家的工作站市场，进一步扩展到消费性电子产品市场，如个人数字助手(PDA)、电子翻译器、电子游戏机，以及交互式有线电视的电视控制盒等，为此组织了一个开发小组。开发小组的方向是希望能够建立分布式的系统结构，同时将软件上的各种新技术移植到消费性电子产品上。为了这个方向，在 1991 年 4 月 8 日，成立了“green”小组，正式展开整个研制计划。为了达到与软件平台无关的特性，以及大部分编程人员能很快熟悉， Gosling 一开始就试着从 C++入手，增加 C++编译器的功能，然而在经过一段时间的努力之后，由于 C++离 green 小组的目标差距太大，最后只好自己重新定义一套全新的语言和系统。于是 Java 的前身—— Oak 出现了。

为了能将整个系统作一次综合性的演示， green 小组开始了“*7”计划，除了建立 Oak 语言的开发工具外，还有 Green Os，用户界面以及硬件部分，最后的目的是做出与现在的 PDA 类似的电子产品。

1992 年 10 月 1 日， Green 小组正式升级成 First Person 公司。与此同时， First Person 得知 Time-Warner 有线电视公司打算开发交互式电视，认为 Java 非常适合应用到这个领域，于是全力投入了这项技术的开发。很可惜，最后因为商业上的某些原因，并未获得 Time-Warner 公司的青睐， Java 没有运用到交互式有线电视的电视盒上。大约在 1994 年年中，全球信息网(WWW)的势头在 Internet 上愈来愈猛，在积累了过去的开发经验，以及检查整个 WWW 的结构以后，开发小组产生了新的想法。过去 Java 的主要目标(与平台无关，系统的可靠性、安全性等)都非常合适 WWW 世界，而且与其它浏览器不一样的是，利用 java 做出来的 WWW 浏览器，可以做到一般浏览器做不到的功能。于是第一个可以在 WWW 上执行的 Java 程序的 WWW 浏览器出现了，这个 WWW 浏览器一开始被命名为 Web Runner，也就是后来的 HotJava 。

Java 和 HotJava 的出现改变了人们对 WWW 的看法。在 Java 出现前，人们对 WWW 上所传输的数据，只不过将它们想成是一页页的文档，而这份文档除了文字以外，还可以放上一些图片等多种数据， WWW 浏览器不过是一个看文档的工具程序罢了， Java 出现以后， WWW 所传输的数据由单纯的文件摇身一变成为了一个个可执行的程序，并且 WWW 浏览器也成为执行这些程序的系统。

在经过 Sun 公司内部一系列的评估之后，在 1995 年 5 月 23 日， Sun 公司终于正式推出了 Java 和 HotJava 这两项产品，很快就引起了工业界的注意。几个月后，当 Netscape 和 Sun 公司合作，在 Netscape Navigator WWW 浏览器中支持 Java 后， Java 在 WWW 上站稳了脚跟。

1.3 Java 的特点

1.3.1 面向对象

和目前很多软件开发工具一样， Java 的第一个特点就是它是一套面向对象的编程语言。以比较严谨的态度来看，一套面向对象的工具至少应该包括下面四个特点：

封装性： 必须有模块化的性质以及信息隐藏的能力；

多态性： 不同的对象对同一种信息，可以按照对象本身的性质加以回应；

继承性： 可以定义一套对象之间的层次关系，下层的对象继承了上层对象的特征，籍此可以实现代码重复利用，并且有效地组织整个程序；

动态联编： 一个对象一旦生成以后，要使用这个对象只需简单的把信息传递给它，不再需要去参考对象当初设计时的规格。只有在程序执行时，才会真正锁定需要的对象，这样的方式可以使程序设计具有最大灵活性。

事实上，如果以上面四个特点去衡量现有的一些编程语言或工具，有很多都不能算是完全的面向对象。如 Visual Basic 缺乏数据的封装性，而 C++并没有办法做到动态联编。但是 Java 在这四点上都可以做得很好。

1.3.2 操作平台无关性

如果要让程序可以在不同的机器及操作系统上执行，那么首先在编写程序源代码时，必须让自己的程序不使用编程语言中有编译器或平台本身定义的功能，譬如说在 C 语言中的 int 整数类型并没有实际定义它的长度是 2 个字节或 4 个字节，必须由在哪种机器或操作系统下执行才能决定。如果我们在使用时，只把它想成是 2 个字节长度的数据空间，那么当这个程序移植到别的平台时，就很难保证不会发生问题。因此，编程语言严格的定义，是确保程序可以在各种平台上工作的第一步。在 Java 的语言定义，我们看不到任何取决工作平台或编译器的功能或特性。

另外一个最大的问题是，各个机器都有不同的低级汇编语言，在这个机器上编译好的机器码，肯定不能拿到其它机器上使用，即使在同一台机器上的机器码(binary code)，由于与程序的执行过程和操作系统信息相关，只要是不同的操作系统，程序编译好的机器码往往也有很大差异，因而不能互用，必须重做一次编译工作。

譬如说在同一平台 PC 下， Xenix 操作系统的程序机器码，不能直接给 DOS 使用。为了解决这个难题， Java 的策略是采取半编译、半解释的方式，定义出 Java 自己的虚拟机。

一、严格的语言定义

为了确保 Java 不受各种平台的限制，在 Java 的语言定义中，所有部分都是经过严格定义的。每个部分都是确定的，所以不管你所使用的机器及使用的编译器如何不同，最后出来的目标码(Object Code)都不会不同。

以基本的数据类型做例子，可以看到每一个数据类型的长度及表示的方式都是确定的(如表 1.1 所示)。因为表中数据类型都有标准的定义，程序设计者在使用这些数据类型

时，也就不会出现意义上的混淆。

表 1.1 Java 数据类型的长度与表示方式

数据类型	数据长度	数据表示方式
byte	8 位	2 的补码(two's complement)
short	16 位	2 的补码
int	32 位	2 的补码
long	64 位	2 的补码
float	32 位	IEEE754 浮点数标准
double	64 位	IEEE754 浮点数标准
char	16 位	单码字符集(unicode character)

另外一个例子是变量初值的设置。在许多语言中，变量的初值是未定义的，假如你定义一个整数变量 a，则一开始 a 的数值就是所谓的初值。就程序的设计概念来看，初值是什么并不重要，因为变量的初值应该由设计程序的人来决定，而不是有系统或编译器来决定。但是如果你写出下面一段 C 语言程序:

```
int a,b ;
b=a+5 ;
```

一般编译器虽然会给出一些警告信息，但是程序依然可以通过编译并执行。至于程序执行时， a 会是多少，就不是程序设计者所知道的。但是在 Java 语言中，即使像上面的程序，程序设计者依然可知道最后 a 和 b 的值各是多少，因为在 Java 语言定义中对各类型的变量初值，也有完整的定义(如表 1.2 所示):

表 1.2 Java 数据类型变量的初始值

数据类型	byte	short	int	long	float	double	char	boolean	refence
初始值	0	0	0	0L	0.0f	0.0d	'\u0000'	false	null

所以执行上面的程序后， a 是 0 ，而 b 是 5 。

二、 Bytecode 中介结构

Java 解决各机器不同机器码限制的方法是定义出自己的一套虚拟机，以及这套机器上所使用的机器码—— Java Bytecode 。在讨论 Bytecode 之前，我们先来看一个 Java 程序从编译到执行的整个过程，如图 1.2 所示。

看了这个示意图以后，就能了解为什么要执行的 Java 程序是经过半编译、半解释的过程了。一个编好的 Java 程序源代码，先通过 Java 编译器编译，产生出 Java 虚拟机的机器码—— Bytecode ，再经过 Java 解释器将 Bytecode 转换实际使用的机器和操作系统上的机器码去执行。

如此一来，只要实际使用的操作平台上有 Java 解释器，这个操作平台就可以执行各式各样 Java 的程序。而对于程序的开发者而言，不需要担心将来程序在执行时，所使用的操作系统和机器是什么，因为只要完成一个编译工作，做出一份 Bytecode ，各种机器上的 Java 解释器便都可以执行它。

虽然说 Java 虚拟机是以软件解释的方式存在于其它的操作平台上，但我们也可以用软件的方式，来实现原来 Java 虚拟机的设计。Sun 公司目前开发的 Java Chip，就是要将 Java 虚拟机，做成实际的微处理器。

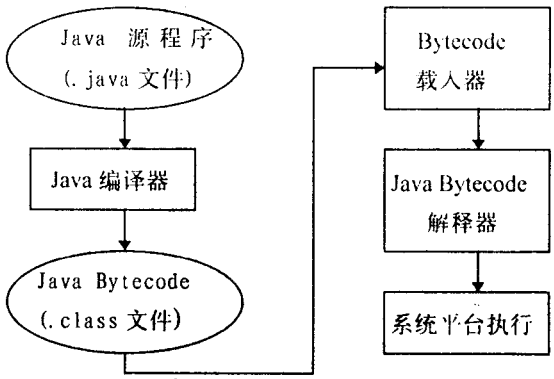

图 1.2 Java 程序执行的流程

三、解释和编译的比较

解释最大的好处是执行时的灵活性，而相对不利之处是执行效率不如直接使用编译出来的机器码。

使用解释的方式可以避免编译时不断的链接和载入动作。这通常是应用程序在开发原型时常用的方式，一旦原型确定了以后，为了求得执行时的最佳效率，再改用的编译的方式产生目标平台上最终的机器码运行。

为了在执行效率和执行上的灵活性之间取得平衡，Java 的 Bytecode 在设计上尽量地接近真正的汇编语言机器码，因此，在解释 Java 的 Bytecode 时，系统可以很快的将 Bytecode 转变成实际的低级指令，取得较高的效率。

然而无论如何，解释的方式终究还是比不上直接执行编译出来的机器码。但是以现在机器的速度而言，这样的效率特性应该是可以接受的。另一方面，由于 Bytecode 的形式很接近实质上的汇编语言，如果对效率的要求十分在意，也可以再开发特定平台上的翻译程序，将 Bytecode 转换成平台上的机器码。同时，因为 Java 的语言格式和 C++非常相似，目前已有人开发 Java 对 C++的翻译器(translator)，进一步将 Java 的源代码翻译成 C++的源代码，再交有 C++的编译器去编译。

1.3.3 安全

Java 这套编程语言既然作为在网络环境上使用的编程语言，就不能不考虑到程序在执行时的安全问题。像不能让计算机病毒等以 Java 小应用程序等方式通过 WWW 传播等。但是什么才是一个真正安全的网络编程语言呢？

Java 对安全性的维护，基本上是以三道关卡来完成的(如图 1.3)，即：Java 语言本身的设计、对 Bytecode 的检查，以及程序执行系统(Runtime System)也就是 Java 解释器的把关。

1. 第一道关卡：Java 语言本身的安全机制

在 Java 语言定义中，对象中的方法和变量的使用限制基本上和 C++是相同的，也就是说，对象中的方法和变量也有 public 、 protected 、 private 等不同的外界使用等级，另外，还可以在类或方法定义时，在前面加上 final 表示该类和方法不能再被其他的类继承使用。这样，对于系统资源有关的对象，就可以利用这种方式，避免被其他对象所修改，或者以继承的方式重新编写。

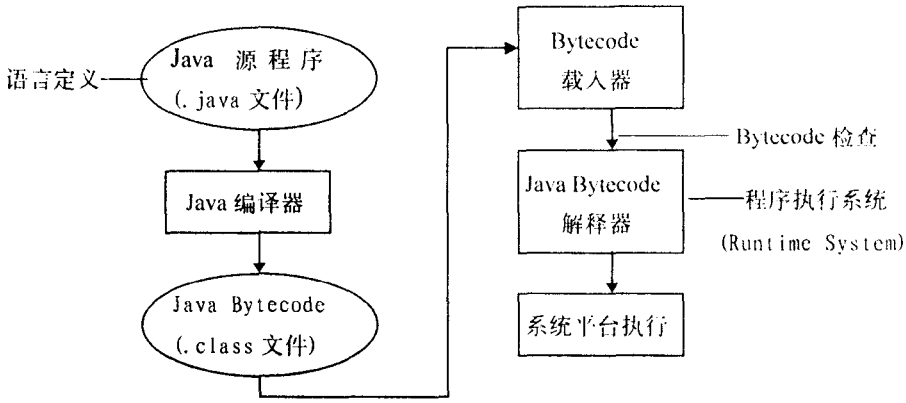

图 1.3 Java 对安全性维护的三道关卡

此外，Java 对于类型的检查，采取程序执行时再检查的方式，可以在程序执行时，确定程序代码不是由别的程序所伪造。而对于内存空间的安全维护方面，Java 取消了 C 和 C++中极重要的指针(pointer),并且把内存空间的垃圾回收(Garbage Collection)工作，交还给系统处理，这样就可以避免有恶意的用户利用指针去改变不属于自己程序的内存空间。

2. 第二道关卡：对 Bytecode 的安全检查

由于 Java 的源代码是编译成 Bytecode 来执行的，因此我们不能因为 Java 语言的安全设计而对 Bytecode 不加检查，因为一个程序的 Bytecode 可能是由 Java 的编译器产生的，但也有可能是人为直接编写的。为了安全起见，在 Java 程序执行系统中负责将 Bytecode 载入的类载入器(Class Loader)也要对 Bytecode 做一次安全性检查，包括确定程序设计没有违反对象的存取权限，使用正确的参数类型来调用对象的方法，以及系统堆栈没有溢出等。

3. 第三道关卡：执行时的安全检查

程序的执行系统(即支持 Java 的 WWW 浏览器)在执行 Java 程序时也要进行安全检查。可以执行 Java 小应用程序的浏览器必须能够阻止 Java 小应用程序对系统资源的不正当使用，以及调整对 Java 小应用程序的处理方式。最极端的情况就是完全禁止 Java 小应用程序的执行。在 Netscape2.0 的安全选项中，就有这样的设计。也因为 Java 小应用程序是通过依附在 WWW 起始页上传送的，所以对于 Java 小应用程序的限制，也比一般的 Java 应用程序严格。

以一般的概念来看，一个安全的网络程序，至少要防止以下几种破坏的可能：

毁灭系统功能：如篡改或删除文件的数据、改变现在使用的内存内容 、终止系统某个进程或线程的任务。Java 通过 WWW 浏览器来制止篡改或删除文件的数据或终止系统某个进程或线程的任务。至于改变现在使用的内存内容， Java 语言对内存空间的权限控制和类载入器对 Bytecode 的检查都可以防止。

消耗系统资源：例如不断地向系统索取内存空间，造成系统内存不足，影响其它程序的执行，或者不断地产生新的进程或线程，排挤正常任务的进程，使其他程序的执行速度变慢等等。为了避免 Java 小应用程序无限制地使用系统资源，在 WWW 浏览器中，可以规定 Java 小应用程序对系统资源使用的上限，一旦 Java 小应用程序超过了原先规定的范围，就停止该程序的执行，防止它影响系统的正常运行。

挖掘系统或个人的秘密：譬如说，将 Unix 的操作系统下的/etc/passwd 用户密码设置文件 E-mail 以及个人或公司的机密，通过网络传给第三者。 Java 除了对存取的极限限制外，也可以通过 WWW 浏览器中的文档或网络的安全检查机制杜绝机密泄漏的可能性。

搔扰正常工作的进行：譬如说，在屏幕产生奇怪的图案，或者让系统不停发出怪叫声。由于 Java 小应用程序是通过 WWW 浏览器的程序执行系统来执行的，所以如果有这样的程序被载入，用户只要通过 WWW 浏览器，关掉程序即可。

虽然说很难保证 Java 的绝对安全，但就一般的标准而言， Java 确实有资格成为一种网络上使用的编程语言。

1.3.4 多线程

随着个人计算机微处理器的飞速发展，个人计算机上的操作系统也纷纷采用多任务和分时设计，将早期只有大型计算机才具有的系统特性，带给了个人计算机。一般可以在同一时期内执行多个程序的操作系统，都有所谓进程的概念。简单地说，一个进程就是一个执行中的程序，而每一个进程都有着自己独立的一块内存空间、一组系统资源。譬如说，每一个在 Windows95 上正在执行的程序，都可以看作是一个进程。

完全不相关的程序在同时执行时，彼此的进程也不会做数据交换工作，它们可以完全独立地运行。但是对同一程序所产生的数个进程，常是因为程序设计者希望能加快整体工作的效率，所以要运用多个进程协同工作。但在进程的概念中，每一个进程的内部数据和状态都是完全独立的，所以即使它们是由一组程序产生的，也必须重复许多数据复制工作，而且在交换彼此数据的时候，也要使用一些进程间通信(IPC)机制。

为了减少不必要的系统负担，线程(thread)的概念也就应运而生。所谓线程，和进程相同的是，它也是一个执行的程序，但是和进程不同的是，多个线程(multithread)是共享一块内存空间和一组系统资源，而线程本身的数据通常只有微处理器的寄存数据，以及一个供程序执行时使用的堆栈。所以系统在产生一个线程，或者在各个线程之间进行切换时，负担要比进程小得多，也因为如此，线程被称为是轻负荷进程(light-weight process)。

虽然使用线程或进程方式来多任务地执行程序有许多好处，而且现在的操作系统也