万水计算机技术实用大全系列

JBuilder 2 实用大全

[美] Eric Armstrong 著

齐舒创作室 译

毅 姚 审校

中国水利水电出版社

内 容 简 介

本书主要阐述了 JBuilder 和 Java 语言的编程思想和编辑技巧。全书通过四部分的内容分别就面向对象的编程基础、Java 的编程元素、高级面向对象编程的思想以及应用程序的构建对 JBuilder 的编程进行了精辟的介绍。书中所附的编程举例使读者能够很快地领会各个编程要点,从而为深入地掌握这门"超级"语言打下坚实的基础。

本书适合于初学 Java 编程以及那些已经具有计算机编程经验的人员开发 Windows 应用程序。

北京市版权局著作权合同登记号:图字 01-98-2021

图书在版编目(CIP)数据

JBuilder 2 实用大全/(美)阿姆斯特朗(Armstrong, E.)著;齐舒创作室译. -北京:中国水利水电出版社,1999.5

(万水计算机技术实用大全系列)

书名原文: JBuilder 2 Bible

ISBN 7-5084-0013-5

Ⅰ.J… Ⅱ.① 阿… ②齐…Ⅲ. Java 语言-程序设计 Ⅳ.TP312

中国版本图书馆 CIP 数据核字(1999) 第 09363 号

书　　名	**JBuilder 2 实用大全**
作　　者	[美]Eric Armstrong 著
译　　者	齐舒创作室
审　　校	毅姚
出版、发行	中国水利水电出版社(北京市三里河路 6 号 100044) 网址:www.waterpub.com.cn E-mail: sale@waterpub.com.cn 电话:(010)63202266(总机)、68331835(发行部)
经　　售	全国各地新华书店
排　　版	北京万水电子信息有限公司
印　　刷	北京天竺颖华印刷厂
规　　格	787×1092 毫米 16 开本 40.5 印张 896 千字
版　　次	1999 年 5 月第一版 1999 年 5 月北京第一次印刷
印　　数	0001—4000 册
定　　价	**75.00** 元(含光盘)

凡购买我社图书,如有缺页、倒页、脱页者,本社发行部负责调换

译 者 的 话

众所周知，计算机领域的发展是任何其他领域所不能比拟的，与此同时计算机编程语言也得到了迅速的发展。不但传统编程语言得到了很大程度上的增强，而且随着面向对象编程思想的不断成熟和各种技术问题的提出，新的语言也层出不穷。两三年前，“Java”这个名词对于大多数编程人员来说还是十分陌生的，更不用说对于一般的计算机使用者了。

Java 是 1995 年 6 月由 Sun Microsystems 公司提出的一种革命化语言，与其他编程语言一样，这种语言在短短的时间内得到了迅速的发展。由于这种语言的易用性、平台无关性、易移植性等诸多特征，使得这门语言得到了广泛的应用。而且，这种语言具有很好的发展前景。为了减少应用程序的开发费用，提高程序的开发效率，使 Java 拥有很好的用户界面和强大的开发工具是非常必要的。Inprise 公司使用最好的 Borland JBuilder 达到了这个目的。JBuilder 具有广泛的代码浏览工具以及用于创建和修改 JavaBean 组件的工具。它还提供了基于组件编程的特色，而且，JBuilder 的环境还可被广泛定制，使得能够更适合于用户的要求。

本书是一本介绍 Java 及 JBuilder 编程的优秀图书。全书通过四部分的内容分别就面向对象的编程基础、Java 的编程元素、高级面向对象编程的思想以及应用程序的构建对 Jbuilder 的编程进行了介绍。书中对作为应用程序组成部分的各个环节的实现方法进行了详细的介绍。丰富的例证，详实的讲解，使读者很快就能进入角色，从而充分体会 JBuilder 的各种编程要点和编程特色，为自己利用 JBuilder 编写应用程序准备了充实的原始资料。总之，在这本书的翻译过程中，译者们充分体会到了这种编程语言的强大威力。我们深信，这本书很值得读者一读，从中必然会受益匪浅。

本书由齐舒创作室翻译。

由于译者使用 Java 编程的时间较短，对语言的深层涵义体会较浅，加之时间仓促，所以书中难免存在疏漏之处，请读者提出宝贵意见。

译 者

1999 年 5 月

致　　谢

在本书中，我与大家分享了通过多年的编程经验获得的编程知识。此书提供了许多关于编程风格的课程，这也是我多年积累下来的。实质上，大多数课程都来自于对人文方面的关心。即对系统最终用户的关心，对将来要修复或扩展程序的人员的关心（这种人可能会是我自己，如果我忘记了怎样做程序工作的话）。

然而对面向对象编程的思考代表了一个更新的面貌。有许多人帮助我转向面向对象的领域。在我与这些人的交往中，发现许多对面向对象程序如何工作的基本了解表面上从未见诸文字。我已经在此书中捕获到这些思想，因而编程人员可缩短自己的学习历程。特别地，我应感谢以下各位。

- Matt Welsh，是他的友好合作精神及帮助使我对面向对象设计及 JBuilder 环境有所理解。对于他离开 Inprise 我感到非常惋惜。
- Steve Abell，是他帮我学到全新的面向对象思考方式，以及在他的站点（http//www.bRising.com）提供了网络/数据库示例。
- Don Chin，来自 Design Patterns Study Group，向我提供了一些本质观点，为我澄清了 Java 适配器使用模式的使用。
- Allen Holub，帮我澄清了 UML 表示法，并带我深入了解了规范化对象设计。
- Alex Tsirfa，帮我思索出编写这本面向对象编程的书。
- Chris Long，来自 KL Group，帮助我理解 JCOutliner 控件。
- Andrew Bennett，来自 Inprise，向我指出了一些我所忽略的 JBuilder 环境的特色。
- 来自 Inprise 的许多友好的开发人员及来自 Sun 的 Swing 开发组，是他们给我分享了许多观点和信息，更为重要的是，他们已经设计开发出一些真正的好软件。

另外，如果没有 IDG BooksWorldwide 的许多友善的职业人士的努力，本书恐怕难以达到今天的效果。我应感谢其中一些人员：

- Matt Lusher，帮助我设计并编写了一本真正符合读者需要的书。他是一个真正的能人，与他一起工作也有乐趣。他在设计和开发本书的过程中一直是一个积极的合作者。每位作者能和这样一位编辑工作都感到很荣幸。
- Laura McCarthy 对大大改进本书提出了许多有见解的意见和建议。她锐利的目光发现许多错误，无疑，本书中仍然存在的错误是由作者造成的。
- Jeff Bankston，帮我处理编辑方面的技术。

所有这些人都对本书提供了可贵的支持。然而，不用说，我已将他们许多好的建议写进了书中，因此凡是在我表达他们建议过程中出现的错误都是我自己造成的。

另外，还应感谢那些为本书设计图形的人。我的图形设计能力非常有限。有一次，在我

成就的高峰时，曾成功地画出一根棒子。因此大家可以明白我为什么要感谢下列人员。

- Jesse Coleman，设计了插图。
- Trevor Wilson，帮助我完成了我所画的不满意的插图。
- Dina Quan，设计了布局。
- Ritchie Durdin，带领图形组完成出版这本精美的书。

最后，有许多人帮我学习并到达今天的水平。实际上，如果没有他们，这本书的出版也将是不可能的：

- Jennifer Atkinson，给我提供了这个机会，是我的好朋友，并信任我；Michael Hoffman，帮我有勇气探索 Java。
- Dava Smith，来自 Dyserv，通过实例教会我对人们努力创造的价值的真正尊重。自那以后，那堂课指导我的界面和应用程序设计。
- Peter Van Der Linden，是他鼓舞并帮助我认识到我拥有可以分享的一些有价值的东西。
- Ellen "Lefty"Leverenz，当我在 Oracle 时，他和我谈论过许多关于写作风格的话题，并向我介绍过 Joseph Williams 的书"Ten Lessons in Clarity and Grace"。这本书向我概要介绍了风格艺术以及如何在清晰表达中利用这些风格。
- Michael Hartstein 和 Ken Jacobs，他们帮我改过许多我在 Oracle 时的早期手稿。他们认真的批评使我粗糙的写作本领成为真正的写作才能。
- 还有许多其他在这些多年中帮助过我的人，包括 Gary Albitz、Stan Jones、Joe Gerard 及 Don Gilchrist。

通过与这些人的联系，我已经学到了许多东西并极快地成长起来。我以衷心的感谢表达他们对我的帮助。非常感谢！

前　　言

此书将带领读者进行一次关于 JBuilder 和 Java 语言的愉快旅行，并帮助读者进入 Java 编程的境界！读者也能学到关于使用 Java 面向对象编程、用户界面开发及基本的编程技术。

本书读者对象

正如本书所演示的那样，几乎任何人都可以使用像 JBuilder 这样的图形 GUI 设计环境完成基于组件的编程。基于这个认识，本书将对 Java 语言进行更深的探讨，从而使读者对 Java 开发可能性有一个整体了解。

本书适合于自学或用作介绍性编程课程中的材料。如果读者对编程有更多一般性的了解，那么本书将变得更容易。但对于那些没有任何编程经验的人来说，也可以使用本书。

本 书 宗 旨

本书期望达到以下目的：

- 使学习容易
- 教会一些有用的模式
- 提供有用的参考
- 教会用户界面编程
- 教会面向对象编程
- 提交有用的程序

使学习语言的过程变得轻松

学习一门新语言的伟大策略是发现一些想要构建的东西并使用该语言来构建它。读者应该挑选一个足够大以获得兴趣但又足够小以便易于达到目的的工程。

本书通过提供一些有趣的例子及练习并为某个有用的目的构建它们，以帮助读者进行学习。在初始的学习阶段可使用本书中的练习，但以后应尝试将它们应用到自己设计的工程中去，以便将这些编程技术变为读者自己的东西。做一遍这个练习时，学到了有关知识，但做 20 遍时，就可直接从脑子里完成。

最重要的是确保该工程是读者真正想构建的东西。这种渴望心情将会在受到阻力或不能解决问题时，甚至于当发现必须重建该程序 90% 的内容时渡过难关。这个动机是取得成

功的基本前提条件。

教授有用的模式

尝试在一个人工智能应用程序中做一些有用的工作会导致对人脑更深的理解。人脑是一部非常优秀的模式识别机器。有一派观点说，最好的教学方法是通过使用例子，这样学者能在头脑中形成对这些模式的抽象概念。

然而，模式并不必总是由例子来感知的，它们也可教会。在这十年中，在规范化模式概念领域已进行过大量的研究工作。面向对象的设计模式、可重复利用的模板及构架都是近来工作的主题。

尽管对设计模式的充分讨论已超出本书的范围，本书后面几章中还是对一些设计模式进行了讨论。另外，本书包括许多有用的代码模式——它们是小段的代码，读者可将它们插到自己的程序中以使用 Java 的某种特色。读者也可扩展 JBuilder 的环境，方法是以代码段（读者可定制并在日后重复使用的类定义模板）的形式定义代码模式。也可学习编程的模式，包括使用 GUI 组件及开发模式。

本书本身也有意于作为一种开发模式服务于读者，它是指使用渐进优化的技术来开发小型到中型应用程序的过程。这个策略意味着开始编写十分小的程序，然后逐步给该程序添加新的特征。使用渐进优化的技术，作者本人曾构建过一个在国际领域排名第五的人工智能程序。该程序花了两年的业余时间。该技术确实管用！

渐进优化对于学习一门语言来说也是一种有价值的技术。它允许读者在最少的时间内做出一些有效的东西，然后发现语言中更多东西时向它添加内容。最近曾统计过，作者本人曾学习过 20 门不同的编程语言以及同样数量的脚本和命令语言，都是使用这种开始完成少量工作然后每次添加一些东西的方法。

提供有用的参考

除了提供一个培训介绍外，本书还设计成一本可用的参考书。包含详细信息的表格阐述了编程可能性的范围。一个或两个最常用的可能途径都在例子和练习中阐述过，这对于学习通用模式足够了。一个关于详细表格的索引在本书开始部分列出，这使得读者日后更易于找到它们。本书后面还提供了一个广泛的词汇表以及关于更多信息的资源列表。

教授用户界面编程

学习怎样设计和构建一个较好的用户界面本身就是一门特长。本书对这个主题进行蜻蜓点水式的探讨，但它将给读者提供有帮助的观点，让读者学会怎样考虑设计过程。

教授面向对象编程

本书帮助读者学习关于面向对象编程的知识。使用渐进优化方法，读者可以不断向应用程序中添加新特征。一个设计得很好的面向对象程序使修改更加容易，因为修改产生的影响仅限于某几个对象的内部。当添加不那么容易时，基本的重新设计过程将变得必要以

便简化代码。本书向读者提供了面向对象的思考方式,可以帮助读者认识必要的重新设计的意义。

提供一些有用的类

本书配套光盘中包含有用于构建和操作层次化数据结构(树)(它们不依赖于包含其中的数据)的完整的类定义和方法。当读者需要创建层次化结构(例如,目录列表)时,这些类使工作更加容易。

有两个主要的工具类

TreeNode:包含树形结构链接,以及用于添加节点、移动节点及列举子树的方法。

Tree:使用 TreeNode 创建和管理数据树。Tree 提供了一些方法,可用于遍历小树,维护可定位"光标",修改光标位置。移动当前行节点以及向当前位置添加节点。

这些类提供了一个构建丰富数据结构的功能以及一些强大的操作工具。

提交一个有用的程序

每一个产品开发者或作者都需回答下列问题:"什么时侯能完成?"一个相关的问题是:"我是否仍然要按日程安排进行?"

奇怪的是看来当前可以利用一个大约 600 美元的工程管理程序来帮助回答这些问题。这样的程序允许编程人员为任务分配一个时段,分配负责人员,跟踪完成百分比,申明任务附属工作,以及打印显示出何时所有工作将完成的美丽图表。实际上,读者是在完成一个单人完成的任务,所做的每一件事都需按顺序进行。就个人日程安排来说,这个昂贵的程序的 90% 费用都浪费掉了。

在另一端是一些"个人组织"程序,它们允许编程人员创建一个任务列表并在日历中保留约会安排。但这些程序中的"任务"不能被分配一个时间段,因此在预测一个工程的完成日期方面没有什么用处。

典型情况是,要创建(有些费力的)一个初始日程表,列出需要完成的任务,估计各自完成需要的时间,并将结果相加,最后标记一个日历。但随着工程的进展,新的任务添加进来,一些任务被更新分配。某些任务通常按与初始预计不同的顺序完成。这时如果读者仍在浏览日程表,要知道进展是十分困难的——特别是在工程中某个关键时间点特别需要知道进展的这一情况时。

本书所开发的一个程序帮助读者回答了这些问题:"什么时候完成?"和"我是否按日程安排进行?"该程序提供了一个层次结构化任务列表。每个任务都有一个时间段,它们都被插放在一个工作日程安排日历中,以便确定提交日期。

本书内容

《JBuilder 2 实用大全》被设计成一本有效的教学和参考工具书。下面是本书的组织方法:

第一部分:面向对象程序设计基础

这一部分介绍了面向对象编程(OOP)的原理以及关于这些原理如何在 Java 语言中实施的内容。另外,向读者介绍了 JBuilder 环境以及构建简单的工程。

第二部分:学习 Java

这一部分介绍了 Java 语言的基本内容,并向读者介绍了使用各种工程进行用户界面编程。这样建立了一个基础,读者可以用它阅读和编写 Java 代码。

第三部分:高级面向对象的思考

这部分对面向对象的方法进行了更详细的探索。读者将学会怎样处理设计一个面向对象应用程序的任务,如何使用 JBuilder 环境的高级特征来开发和提交应用程序。

第四部分:创建应用程序

这部分处理了两个中型工程:Othello 游戏和 Scheduler。读者将探索更详细的设计和开发过程,并学到更多有关 Java 平台的特色。这些工程给读者进行以后的探索和试验提供了一个基础。

第五部分:附录

这些附录提供了附加的信息、参考资料以及附加资源指南。本书配套光盘的内容(包括工程及参考文件列表)也在这里进行了描述。

本书约定

本书中每一章都包含一些标准部分以帮助读者学习该章中介绍的材料。

本章要点:这一部分列出了该章中介绍的主要内容。

附加说明:附加说明包含该章主要材料以外的有用信息。

小结:在每章末尾都列出了本章所介绍材料的概要。

另外,有几个图标用来说明不同类型的材料:

这些技术细节通常是给有经验的编程人员看的。如果读者是初学者,浏览一下就够了。头脑中形成一个大概的印象,不必担心未理解全部内容。

提示提供了一些有用的信息和关于如何完成工作的建议。好的编程经验通常是像口头流传下来的民间故事一样传下去。这些部分记下了一些真知灼见。

这里是第一个提示:看一眼键盘就会传递很多惊人的知识。当今,程序是非常复杂的,

编程人员总能学到一些新的东西，然后将这个新的本领展示给其他人看。

下面的字体风格约定有助于帮读者找到文字：

- 黑体：表明需完全无误地输入在一个代码清单中，黑体部分是输入的部分。以普通字体书写的周围代码显示出新代码应放的位置。黑体代码也表明接下来的段落中将对这些代码进行讨论。
- 斜体：表明需读者输入自己的数据，例如："Enter *Your Name*"。
- 单空隔文字：用于显示在屏幕上的信息、文件名、URL、Java 代码以及出现在普通段落中的系统消息，如 this。

与本书作者联系

读者可将自己的建议、问题、赞赏及偶然出现的错误报告发送给本书作者：

E - mail：eric@treeLight.com

Web：http://www.treeLight.com

同样，也可查看 IDG BooksWeb 站点中有关的其他例子、Java 语言的新特征和新内容：

http://www.idgbooks.com

简　　介

Java 的伟大之处

刚开始它是作为一门小巧的用于书写 Web 页面小应用程序的语言，但马上就成为编写大型应用程序的语言了。而且，Java 正变成大型、以网络为中心、与任务有关以及服务器/客户应用程序的开发语言。

由于有强大的错误检测超强功能及网络上有大量的样本代码，Java 比其他语言用起来更具生产力。当发现 Java 的网络功能及 Java 程序能在任何平台上运行的功能后，就更容易明白为什么这门语言如此吸引人。

首先看一下 Java 具有广泛普及性的原因：

- 书写一次，运行于任何平台
- Java 的“虚拟机”技术
- 高性能，面向对象
- 更具可靠性
- 更有趣
- 更易于维护
- 以 Web 为中心
- 广泛被人接受
- 使开发企业级应用程序成为可能

JavaScript 不是 Java。JavaScript 是一门与 HTML 兼容的脚本语言，它具有一个利用 Java 宣传而选择的名字。不巧的是，这个名字易产生混乱，它看起来与 Java 有关，但实际上这两种语言无任何关系。

书写一次，运行于任何平台

使用 Java，在自己系统上书写的程序将会在现有的平台上不用修改就可运行。另外，具有图形用户界面的程序也有本机操作系统的外观。使用新的 Swiny 组件，用户甚至可选择他们更喜欢的外观。这是很酷的。另外，每个主要的操作系统生产商都已承诺对 Java 的集成支持，这表明 Java 程序就像这些平台上其他程序一样地运行。

不巧的是，Microsoft 已经退出了以前的承诺，现在正致力于分割无所不在的 Java 环境。Microsoft 的退出增加了 Sun 的工作量，Sun 除了必须为它们自己的平台开发外，还必须为 Microsoft 平台开发。现在，它们正在做一件了不起的工作，使得

到处都保持着全部的 Java 兼容性。

这种可移植性是相当不错的。但还很难说 Java“书写一次,运行于任何平台”的哲学使得它被管理者及开发者更加喜爱。

对于管理者来说,Java 承诺节省许多费用。在过去,新的机器和新的操作系统需管理者抛弃成千上万条代码,这样可利用更新的技术。另外,管理者不得不接受以新技术对员工进行重新培训的费用。仅有的选择是,使用过时的效率低下的系统,直到没有修改造成的费用对企业产生了副面影响以至不得不做修改时为止。另一方面,Java 承诺将来旧的代码不必舍弃,以利用更快或更新的操作系统。

提供交叉平台应用程序的生产商日子更难过。例如,OradeCorporation 开发了一个数据库,它可在每个主要平台上运行,但完成这个工作需花费巨大的人力。某个部门只负责定义用于编程的“虚拟机”,以及定制用作编写代码的 C 语言。这门语言被大大地定制了,以至于几乎与 C 没有相同之处!即使是一个有经验的开发者也要花上好几个月的时间来学习这种语言变种。即使那样,结果也是不可移植的!另外五个部门将基本应用程序移植到 80 多个不同的 UNIX 及 20 个其他操作系统中。如果每个部门都有 20 到 30 人,大约需 150 人来参加“移植”这个应用程序的工作。

因为 Java 承诺要最小化或完全去掉这样的费用,它对企业管理者具有很强吸引力。Java 对开发者也具吸引力。对于开发者来说,技术改变意味着抛弃成千上万个学习时间,以及抛弃所有积累下来的工具、库及捷径。相比之下,Java 预示将来开发者能利用以前的知识和掌握的库及工具来工作。Java 在软件开发领域比其他语言具有更强大的生产力。

Java 的“虚拟机”技术

或许 Sun 对 Java 做出的最大贡献就是提出并发布了 Java 虚拟机这一技术。虚拟机是 Java 具备移植性的基础。它实际上是一个具备整套特色的抽象计算机的规范。虚拟机已经在几乎每个主要的平台上得到了实现,这一点就是 Java 能“在任何平台上”运行的原因。即使在 Java 被计算机语言中的重大改进所替代之后很长一段时间内,虚拟机将仍然可能是这门新语言运行的基石。

无论如何 Java 并不是一门十分完美的语言(参见第十八章更多信息)。然而,即使一门新语言取代 Java,这门新语言也将无疑会遵循虚拟机的规范。Sun 最宝贵的传家宝将可能是虚拟机标准——即使在 Java 被另一门语言取代之后很长一段时间内也将继续存在。

如果用户对编程不熟悉,可以浏览一下本部分最后一段内容,以获得一些主要的概念。如果用户在其他语言中碰到过这些细节,它们也只能使用户获得一些理性认识。

高性能,面向对象

Java 代表了计算机语言领域中一个少有的进展。与此同时,它还是面向对象的,并具备高性能及可移植等特点。要达到这种平衡状态是非常困难的,但 Java 的开发者们获得了成

功。

Java更加具有面向对象的特点，这不像C++这种“混合型”面向对象语言。C++被限制为保持与现有C程序的向上兼容性，Java可自由地实施纯面向对象的模式。虽然Java程序失去了一些灵活性，它们却拥有更多的可读性和可靠性。结果Java程序更加简洁，更易于阅读和修改，而Java本身更易于学习。

在某些情况下，Java比C++功能更强。例如，在Java中，用户可以以字符串的形式指定任意类的名字，并创建该类的一个对象——这在C++中是做不到的！

但是，Java并不将面向对象的特点运用到极限。在像Smalltalk这样的语言中，即使像整数这样的原始数据类型也是对象。如果用户想给一个整数增1，需要给它传递一条消息来告诉它给自己加一个1。这种方法并不完全差劲(结果是这门语言更具连续性)，但对性能具有影响。

除了整数外，Java将字符和字符串也当作基本数据类型。这样，Java开发者们将纯面向对象和实际需要进行了权衡。他们消除了一个“纯”面向对象方案造成的性能影响，因而消除了可接受性的一层主要障碍(这种性能的影响部分地阻碍了Smalltalk的采纳范围)。但结果这门语言确实变得更加复杂。然而，从总体上来说，开发者们已经愿意处理这种复杂性，以便获得增强的性能及简单任务更“自然”的编码方法。

最后，Java将速度与可移植性进行了权衡。Java源程序被编译成可移植的字节代码，这种字节代码被一个与具体机器相关的翻译器解释执行。这种技术并不是新的。大约在20年前，UCSD Pascal系统就采用过这种技术。现在已具备一些因素，使得这种技术比以前更具生命力。

第一个因素是计算机速度。目前的普通计算机也比20年前的计算机快许多倍。实际上，目前的最低性能的个人计算机也要快许多倍。结果，字节代码解释技术在许多应用程序中具有更强的可接受性，而在过去，运行的速度是非常慢的。例如，具备更多用户交互性的程序需比执行指令花费更多的时间等待用户输入，因而字节代码解释技术对性能的影响几乎是看不出来的。

第二个因素来自编译技术的进步。“Just in Time”编译器可产生优化的本机代码，这些代码在程序执行时存放在内存中。当代码的某个特殊段下次执行时，本机代码被予以利用，因而节省了解释的费用。JavaSoft的“HotSpot”新技术也能解释字节代码，代码内嵌和优化这些优越的特色组合起来，以保证达到可与那些完全编译后的程序相比的速度。

另外，Java是一门多线程的语言。具备可并行操作的多个线程这一能力使得用户可用Java来编写大型、高性能并能有效地使用时间的应用程序(对于实时游戏及仿真程序使用Java也是很方便的)。通过将速度、可移植性及面向对象进行权衡，Java开发者们已经开发出一门具备广泛用户的编程语言，这门语言用于许多应用程序。

更加可靠

Java具有许多特色，这些特色使得它比其他高性能的语言(如C和C++)更加可靠。这些特色包括：

· 编译时错误检测
· 运行时错误检测
· 垃圾回收
· 无指针算术运算
· 安全检查

编译时错误检测

编译时错误检测的最大优点是在产品发布之前可保证每一个编译时错误都被发现。因此,在编译时发现错误会更为合理。Java 使用这种方法完成了一项很好的工作。

Java 编译器在编译时会发现许多能使运行时程序出现问题的错误。笔者最爱出现的错误是"Variable may not be initialized(变量可能没有被初始化)"。在编写代码的忙碌工作中,很容易忘记初始化某个变量或仅在一条路径上对其进行了初始化。例如,看看下面代码段:

```
if (x==1) {
  r = 2;
}
print (r*10);
```

在此例中,编译器注意到 r 可能未初始化,因而给用户提供一个警告,非常棒!

垃圾回收

大多数语言都不具备自动垃圾回收功能。结果,当内存被分配但未释放时,溢出现象常有发生。假定每次都调用某个例程,它都被分配一块内存。同样假定该例程退出有时不会释放这些内存。一段时间后,越来越多的内存从操作系统中取走而未被回收。最后,由于缺乏内存,整个系统都将崩溃。

内存溢出是发生在运行时的另一类错误。Java 的自动垃圾回收功能可将那些不再使用的内存释放出来。这个特色不仅解除了程序员不得不担心将内存返回给操作系统的烦恼,也能预防一些严重的而且通常是难以检测到的错误的发生。

无指针算术运算

Java 确实具有指针。这个事实是十分重要的,因为指针是构建许多强大数据结构的基础,同时也是提高性能的核心。然而,在 Java 中,指针是在对象引用中继承过来的。在程序中指针并不可见,而且最为重要的是,它们不能使用算术函数来操作。

这种方法对于 C 和 C++ 程序员来说是一个很好的消息,因为这些程序员需经常对作为参数传递的指针进行引用。然后,他们需解除对参数的引用才能获得指针,以及解除对指针的引用才能获得所指向的对象。听起来似乎有点复杂? 确实如此。如果编程人员一次解除的引用太多或太少,结果会造成某个指针指向一些任意的内存区段。读取或写入到错误的内存区段会造成另外很难发现的运行时错误。

当编程员操作指针时,还会发生其他的错误。编程员或许要给指针赋值以获取下一个数据结构,但是,并不能保证已赋了正确的值或者数据将占用的内存位置还未得到释放。

Java 消除了可视指针及指针算术运算，这一点预防了许多可能发生的运行时错误。

运行时错误检测

有些错误只有直到程序运行时才能被发现，它们仅在某些条件下发生。但尽可能把 Java 错误定位到原发位置。这种方法意味着编程员能发现发生了错误的地方，而不会造成程序中一个较远位置发生一些其他（表面上并不相关）错误。例如，Java 对字符串和数组索引进行运行时的边界检查。

假定有一个含有 10 个元素的数组。在未受保护的语言（如 C）中，编程员需写 11 个元素，读取 11 个元素，然后那一部分程序才会工作正常。然而，通过写 11 个元素，编程员可能会覆盖掉其他内容，修改此值会在运行较晚而且运行次数较少的代码中产生错误。发现这些错误是一项十分困难的任务。Java 使得发现错误更加容易，因为这些问题大多发生在错误最早产生的地方，而不会在另外一段不相关的代码中发生。这真是太“酷”了！

安全检查

在应用程序运行之前，字节代码解释器将有意或无意地校验该程序是否已损坏。从网络上下载的小应用程序不仅会通过这个严格的校验步骤，也会被限制修改局部计算机系统。结果，这些程序值得信任，它们不会造成无意或有意的破坏。因为这种保护措施内嵌在系统低层，实际上不可能绕过这些检查过程。这个措施已经在恶意病毒极易创建的环境中变得日益重要。

更加有趣

Java 更加有趣，因为编程员在构建内容方面所花费的时间更多，而在跟踪难以发现的错误方面花费的时间更少。编程员也将具备更高的生产力，因为 Java 提供了许多使得编程更加容易的特色。像字符串处理方法、异常处理、向量及散列表这些特色给编程员提供了方便的工具，因而编程员与语言本身打交道的时间更多，而与语言不太相关的工作时间更少。

更易于维护

在一个主要工程的生命周期中，往往是将更多的精力和费用花费在维护和扩展应用程序上，而不是在原始程序的开发上。尽管 Java 对这个问题并没有提供完美的解决办法（更多的信息见第十八章），但它具有一些特色，能在某种程度上控制这个问题。

到目前为止，最大的维护费用是程序开发人员在学习应用程序怎样工作方面的投入时间。Java 尽可能地缩短这个时间，方法是改进程序的可读性。程序可读性的增强主要来自于 Java 省掉了下列要素：

- 没有 goto 语句
- 没有操作符重载
- 没有预处理器

没有 goto 语句

尽管 Java 没有 goto 语句，但它具有编程员所需要的分支语句，包括标识语句 break 和

continue 及 try/catch/finally 语句。

当发现程序中有支持 goto 的标识时，编程员可知道程序分支到哪一点，但无法知道是来自哪个位置。程序可以从一些任意位置到达那里。而 Java 对程序分支进行了更多的限制。当编程员看到一个标识时，可知程序仅能通过一个循环语句或标识附加到的其他代码块内的 break 或 continue 语句到达那里。

使用 try/catch/finally 指令更增强了程序的可读性。当看到下面这样的代码时：

```
try {
  ...
```

编程员会立即发现闭合括号后面的以下某个地方会有一些异常处理语句，如：

```
catch ( ...) {
  ...
}
```

另外，编程员可能会发现一个语句，如：

```
finally {
  ...
}
```

这表明在那部分的代码总是被执行的，这与 try 块代码怎样退出无关。

这些结构对处理错误及其他“异常”条件，保证完成最后工作的代码总被执行(这些任务即是这些结构的主要目的)是很有用的。另外，它们也增加了程序的可读性。这些代码的结构与功能不用检查就看得很明白，这对于维护来说是相当有益的。相反，使用标识符和 goto 语句，编程员需对程序进行深入的分析才能确切知道它们的目的所在。

没有操作符重载

操作符重载看起来很棒。使用操作符重载，编程员可以对任意数据结构定义加法运算以完成所需要的工作。例如，如有两个数组：array1 和 array2，可以对这些数组定义加法运算，目的是“将 array1 的每个元素加到 array2 相应的元素上”。C + + 提供了这个功能，而且某些圈内人士还挺喜欢这个功能。

然而，这种灵活性也是需要付出代价的。当阅读用允许操作符重载的语言编写的程序时，无法确信看到的操作是编程员所期望的那样。更糟的是，这个特色很少使用，因而编程员易于假设知道这些操作的含义。编程员可能要花费很长时间才能解开一段代码的含义，且最终才发现以前的理解完全是错误的。

没有预处理器

支持预处理器的语言允许编程员定义新的语言要素。在编译之前，预处理器将这些要素翻译成语言能理解的结构，这样可保持程序的可读性。当然，这也丢失了一些灵活性。有时，编程员可能想使用预处理器使工作起来更加方便。然而没有预处理器，任何人都能拿起一个 Java 程序就知道程序的含义(如果这些人接受过 Java 培训的话)。这种方法使“Java 知