北京大学出版社 正宗Java丛书

Java并发程序设计

——设计原理与模式

〔美〕Doug Lea 著
严伟 陈志兰 玄伟剑 赵海燕 等译
陈向群 审校

JAVA

北京大学出版社
艾迪生维斯理出版有限公司

Java 并发程序设计

——设计原理和模式

〔美〕 Doug Lea 著

严 伟 陈志兰
玄伟剑 赵海燕 等译
陈向群 审校

北京大学出版社
北 京

著作权合同登记号　图字：01-97-1091

图书在版编目(CIP)数据

Java 并发程序设计：设计原理和模式/(美)利(Lea,D.)著。－北京：北京大学出版社，1997.8

ISBN 7-301-03479-2

Ⅰ. J…　Ⅱ. 利…　Ⅲ. Java 语言-并发程序-程序设计　Ⅳ. TP312Ja

书　　名：Java 并发程序设计——设计原理和模式

著作责任者：〔美〕Doug Lea 著，严伟等译

责 任 编 辑：沈承凤

标 准 书 号：ISBN 7-301-03479-2/TP・352

出　版　者：北京大学出版社

地　　址：北京市海淀区中关村北京大学校内　100871

电　　话：出版部 62752015　发行部 62559712　编辑部 62752032

排　印　者：北京大学印刷厂

发　行　者：北京大学出版社

经　销　者：新华书店

787×1092　16 开本　15.375 印张　382 千字

1998 年 2 月第一版　1998 年 2 月第一次印刷

定　　价：36.00 元

丛书前言

Java丛书为Java程序员和最终用户提供了权威的参考文档。这套丛书由Java组成员编写，在Javasoft部，即Sun Microsystems的一个业务部的赞助下出版。World Wide Web使Java文档在Internet上可通过下载或作为超文本获得。然而，全世界对Java的关注促使我们编写了这套丛书。

为了解Java下载Java最新公开版本的最新情况，请访问我们的World Wide Web站点http://java.sun.com。要了解有关Java丛书的最新消息，包括实例代码、勘误和新书预告，请访问http://java.sun.som/books/Series。

感谢Addison-Wesley专业出版集团，为出版这套丛书与我们保持了非常好的合作关系。编辑Mike Hendrickson和他的小组在帮助我们出版这套丛书的过程中做了极出色的工作。Sun Microsystems公司的James Gosling，ruth Hennigar和Bill Joy的支持确保本丛书包含了使其畅销所必不可少的内容。一点个人的感谢给我的孩子们：Christopher和James，因为在丛书编写过程中，他们在许多次到我办公室的路上给予我动力。

Lisa friendly
丛书编辑

Library of Congress Cataloging-in-publication Data

Lea, Douglas
Concurrent Programming in Java: Design Principles and Patterns / Doug Lea.
p. cm. — (Addison-Wesley Java Series)
Includes bibliographical references and index.
ISBN 0-201-69581-2 (pbk.)
1. Java (Computer program language) 2. Parallel programming (Computer science) I. Title
II. Series: Java series
QA76.73.J38L4 1996
005.2—dc20

96-43733
CIP

ISBN 0-201-69581-2
2 3 4 5 6 7 8 9 10–MA–00999897
Second printing, January 1997

前　言

Java 程序设计语言席卷世界，像 C++和 Small talk 这样的面向对象程序设计语言日益频繁的公众曝光，为其添加了燃料，它是从 Internet 开发开始的。在众多特征之中，Java 对线程提供了简单而紧密集成的支持，并因而令许多人首次接触到并发程序设计。

并发程序(*concurrent program*)由多个任务组成，表现得好像它们同时在执行。在有多处理器的计算机系统中，可能确实如此——几个任务在各个处理器上同时执行。在只有单个处理器或者任务多于处理器的系统中，系统可以在任务间切换，使得看上去比处理器多的任务同时在执行。

并发程序的任务经常用线程("控制线程"的简写)来实现，指令序列在封闭程序中独立运行。各线程可同其它线程共享对存储器的访问，并可拥有协调其活动与其它线程、计算机系统或用户活动的方法，来达到程序的目的。

编写并发程序需要以超越许多程序员经验的方式看待事物。要获得这种新的观点可能是困难的。许多线程包是复杂的。并且它们的文档可能因此停留在只讲要点的方式上。一些方法是学术的并且相对地不易办到。因而，线程程序设计或多或少仍是一种黑色艺术，由少数试图以某种方法将其全部解决的人来实践。在他们之中，并非都学会了简洁而可靠地使用线程。

尽管 Java 的线程化模式相对简单，编写 Java 并发程序可能仍是陌生的、令人困惑的。大多数处理 Java 语言的书以一章覆盖并发程序设计，介绍 Java 的线程类，但并不给读者以概念背景来用线程高效地编写程序。

这是第一本致力于 Java 并发程序设计的书。Doug Lea 通过像 Design Pattern(Gamma，Helm，Johnson，Vlissides)书中的现已熟知的模式途径，扩展了面向对象的设计示例。通过提供设计高效、可靠和简洁的并发程序所必需的概念结构，他在典型的程序设计经验和各参考资料的鸿沟间架起了桥梁。

当涉及 Java 线程程序设计细节时，Doug 展示给读者如何以一种在使用并发程序设计的其它途径——而不是仅使用 Java 时同样适用的方式来考虑并发程序设计。他为并发程序设计提供了一本实用的设计模式食谱，它对新的线程程序员将是指导性的，非常有用的，而对于有经验的程序员同样如此。

Tim Lindholm
JavaSoft
1996 年 9 月

致　谢

本书源于我在1995年春季汇集在一起的一部分Web页，那时我正试图了解自己对试验性开发中使用Java并发特征的早期尝试的意义。此后它逐渐增长，首先是在World Wide Web上，在那里我扩展、扩充并删除了一些模式来反映我本人及其它人的日益增长的Java并发性经验；现在成为本书，它在并发软件开发原则的远景下安排各种模式。那些网页仍存在，但如今是作为对最适于本书形式的概念展示的补充。

在此过程中已有许多变化，这得益于评论、建议和同许多和蔼而又知识丰富的人的交流。他们包括：Taranov Alexander，Il-Hyung Cho，Bruce Eckel，Ed Locke，Mike Mills，Trevor Morris，Andrew Purshottam，Simon Roberts，Joel Rosi-Schwartz，Aamod Sane，Doug Schmidt，Kevin Shank，Sumana Srinivasan，Henry Story，Satish Subramanian，Jeff Swartz，Patrick Thompson，Volker Turau，Cees Vissar，Bruce Wallace，Greg wilson，和Steve Yen，以及许多提出匿名电子函件方式评论的人。

Ralph Johnson的模式研究小组的成员（特别是Brain Foote和Ion Chai），通读了一些模式的早期形式并提出了许多改进建议。本书手稿的正式和非正式校阅者也作出了许多贡献。他们包括：Chan，Cary Crain，Desmond D'Souza，Time Harrison，David Henderson，David Holmes，Time Lindholm，James Robins，Dreg Travis，Mark Wales，和Deborra Zukowski。特别要感谢Tom Cargill，因其在过去一年中的众多洞察和改进，并且因其允许本书包含对他的指定通知模式（在第8章中）的描述。

在建立索引的过程中，Rosemary Simpson作了许多改进。Ken Arnold耐心地协助我同FrameMaker打交道。Mike Hendrickson和Addison-Wesly的编辑小组给予了持续的资助。

若非Sun实验室的慷慨支持，本书将无法完成。特别感谢Jos Marlowe在有趣而又令人激动的研究和开发项目上的合作。

最重要的是要感谢Kathy，Keith和Colin，因为他们容忍了所有这些。

Doug Lea
1996年9月

目　录

第1章 引 言

目前面向对象(OO)的程序设计语言相对来说还不多见,而 Java 程序设计语言(以下简称"Java")是其中的一种,它吸收了线程及相关的并发结构,而无需特殊工具或支持系统。用 Java 进行并发程序设计比其它大多数语言更容易也更自然。

1.1 并发性的应用

本书讨论了用 Java 考虑、设计和实现多线程代码的不同方式。但在阅读如何做这些事情前,读者可能首先会考虑为什么以及何时使用并发设计。

1.1.1 优点

并发性开辟了在顺序程序中无法实现的设计可能性。线程将程序员从那种调用一个方法,然后阻塞,在等待响应时不作任何事的代码局限中解放出来。使用线程,可以另外触发能并发运行的新的独立事件,等待或不等待它们结束。开发线程的理由包括:

反应程序设计(Reactive programming) 一些程序需要同时做一件以上的事,以此作为对一些输入的响应。例如,World Wide Web 浏览器可能同时执行 http GET 请求获取 Web 页,播放声音片段、显示一些图像的接收字节数及忙于某个用户的建议对话框。虽然可以用单线程方式通过人工交叉执行不同活动来设计这样的系统,但很复杂,脆弱,并且容易出错。使用线程更容易设计和实现反应程序。事实上,本书描述的大多数设计原理与模式,适用于反应程序。

可用性(Availability) 并发性允许程序员维护高度的服务可用性。例如,在较通用的并发设计模式(见大多数 Internet 服务,甚至许多小应用程序(applets)中,使一个对象充当服务的网关接口,通过构造新线程异步执行关联的动作来处理每个请求。同时网关可以快速接收另一个请求。这样有助于通过疏通未决消息的通信网来避免瓶颈问题。它还可以提高访问的公平性:新的可快速服务的请求不必等待老的耗时的请求结束。

可控性(Controllability) 线程内的活动可被其它对象挂起、继续和停止。它提供了在顺序程序设计中所没有的简单性和灵活性,在顺序程序设计中要停止一个活动(暂时或永远)并做其它事的愿望常常是难以实现的。

主动对象(Active Objects) 软件对象常常以真实对象为模型。大多数真实对象显示出独立、自主的行为。至少在某些情况下,实现这种设计最容易的方法是无论何时创建这样的对象,都启动一个新的线程。

异步消息(Asynchronous messages) 当一个对象传送信息给另一个对象时,不必总关心动作何时执行。线程允许第一个对象继续自己的活动,而无需等待无关动作的结束。

并行(Parallelism) 在一台具有多 CPU 的机器上,并发程序设计可用来开发可用的计算能力来提高性能。即使没有多 CPU,线程中的交叉活动也可避免同耗时处理相关联的延迟,这

种耗时处理不必在其它活动启动前结束。

并发需求(Required Concurrency) 即使不是明确地要编写并发程序,许多预定义的 Java 支持类和运行期特征以并发方式操作。包括播放声音片段和显示图像的 java.applet 和 java.awt 类,以及导致每个 Java Applet 在自己的线程中运行的机制。

1.1.2 限制

如果并发是卓越的,那么程序员应处处使用它。但不该这样。并发的好处必须同其在资源消耗、效率和程序复杂性方面的代价比较:

安全性(Safety) 当多线程不完全独立时,每个线程可能需要对象发送消息给另一个对象,而那个对象可能涉及其它线程。所有这些对象必须利用同步机制或结构互斥技术来确保它们维护一致的状态。如果试图使用的多线程仅涉及到在顺序设置中工作的对象,则可能导致随机查看、难于调试的不一致性。另一方面,同步机制可能增加程序的复杂性。

活性(Liveness) 并发程序内的活动可能没有活性。即一个或多个活动可以因为一些原因中的任何一个简单地停止;例如因为其它活动正使用全部 CPU 周期,或因为两个不同活动死锁(deadlock),都在无限期互相等待。

非确定性(Nondeterminism) 多线程活动可以任意交叉。同一程序的两次执行不必相同。需要大量计算的活动可能在那些实际上不需要计算的活动之前结束。这可能使多线程程序更难预测、理解和调试。

线程与方法调用(Threads versus method calls) 线程对于请求/应答方式的程序设计不是很有用。当一个对象逻辑上必须等待来自另一对象的应答以便继续时,同一线程应被用于实现整个的请求—执行—应答序列。当没有用于并发的空间时,构造新线程将得不到好处。相反,作为线程运动的活动不能使用标准调用方式,在这种方式中客户发送自变量,等待它们被处理,然后接收应答结果。这些效果可在线程中获得,但需要特殊编码。

对象与活动(Objects versus activities) 基本上在所有面向对象系统中,异步执行的并发活动的数目比起对象数目略多一些。即使从主动对象来看,仅当调用实际上产生新的异步活动时,创建新线程才是有意义的。并非无论何时都盲目地、无条件地自动创建新对象,而不管这个新对象有可能参与异步活动还是永远不可能参与异步活动。

线程构造开销(Thread Construction Overhead) 构造线程并在运动中设置它,通常比构造正常对象或在其上调用方法更缓慢,存储需求也更大。如果活动只是少数几条基本语句,则通过方法调用来调用它比使用线程要快得多。

上下文切换开销(Context-switching overhead) 当活跃线程比 CPU 更多时,Java 运行期间系统经常从运行一个活动切换到运行另一个活动,它还需要调度(scheduling)——了解下一次运行哪个线程。

同步开销(Synchronization overhead) Java 实施同步方法可能比那些不提供适当并发保护的语言慢。而那些必须依赖于对象的当前状态以推迟和继续动作的方法可能开销更大。在线程和同步开销之间,并发程序可能比顺序程序运行得更慢,除非有多个 CPU,而有时即使如此也是一样。

线程与进程(Threads versus processes) 本质上自包含、足够重的活动可以更简单地封装为独立程序。独立程序可由系统级(并发)执行设备或远程调用机制访问,而不是作为一个进

程的多线程部件。(虽然界线是模糊的,进程通常定义为维护它自身的资源集的活动实体,而线程使用包含该线程自己的那个进程的资源。)

1.2 概　述

并发引入了在顺序 OO 程序设计中没有的设计以及程序设计机遇和问题。本书主要通过重用已证明是对通常并发 OO 设计问题有效解决的构造和技术,描述一些开发这种机遇和解决问题的方法。

未集成到程序设计语言中的线程包往往难于使用,臃肿的手册常常只用于了解如何表达简单的构造。Java 不是这样。本章的后面部分考察几个同并发程序设计特别有关的 Java 构造。(假定你熟悉 Java 程序设计的其它方面。)本书其余部分集中于这些构造在类、部件、框架和应用程序的设计中的角色和使用。

本书是关于设计,而不是关于特定的并发算法或其正规分析的。已有许多介绍在 Java 中实现算法的好资料(特别是在深入阅读中列出的由 Andrews 所写的书)。相反,本书收集了来自非 OO 并发程序的标准设计技术,被证明在其它并发程序设计语言中有用的构造,来自有关 OO 并发性研究文献的新思想以及来自实际的软件开发中并发应用程序的实际的考虑。并以在构造并发 Java 部件、applets 和应用程序时使用或重用的方式来介绍它们。

一些概念是以这样的方式引入的,首先介绍在引入这些思想的语言和系统中构造的有混合语言特征的 Java 等价体将成为什么样子,然后介绍如何在 Java 本身中获得这些效果。这样就提供了设计并发部件的各种概念性工具。也使你更容易适应这里未包括的,源于其它语言或其它途径的并发技术。

设计模式(design patterns)有助于组织可用于结构化并发 Java 程序的技术。模式描述一种设计形式,通常为对象结构(object structure)(也称为微体系结构(micro-architecture)),而对象结构由一个或更多遵循某种静态和动态约束和关系的接口、类、和/或对象组成。模式是描述这样一些设计和技术的理想工具,它们不需要以完全相同的方式跨不同上下文实现,并因而不能有用地封闭为可重用部件。可重用部件和框架可以在软件开发中充当中心角色。但并发 OO 程序设计中有许多是由循环设计模式和惯例的重用、适配和扩展组成的,而不是由特定的类组成的。

基于模式的方法还可以帮助在并发程序设计的理论和实践间不幸存在的间隙间架起桥梁。有关并发性的研究有时依赖于对日常 OO 软件开发不合适的模型和技术。另一个极端,一些基于线程的程序是轻率设计的结果。本书试图取一中间地带,从各处"窃取"好的思想和最好的经验并将它们整理为易于应用的形式。

这里使用的大多数术语和记号,采自 Gamma,Helm,Johnson 和 Vlissides 所著的 *Design Patterns* 这本先驱著作。这里介绍的一些范例是普通顺序 OO 设计模式对于并发程序设计问题的扩展和应用。特别地,*Design Patterns* 中给出的大多数模式都被本书使用或引用。这些模式都在第一次出现时给出概述,虽然为防止涉及细节只作了很简洁的说明。如果尚未读过 *Design Patterns* 可以忽略这些引用。

与 *Design Patterns* 中模式不同的是,这里的模式嵌入到相关上下文及软件设计原理讨论的章节中,而在这些软件设计原理的模式中起到了主要的影响和关键的限制。这种展示风格有

助于组织令人迷惑的各种模式，这些模式范围从小的常用程序设计构造到应用程序级的结构化技术。此外，本书坚持以基于模式的处理方式，澄清设计形式、上下文、可应用性和结果的基础规范。它以建设性的、配方似的方式描述解决方案。事实上，许多模式展示对方结构外，还包括设计步骤(design steps)，因而比 *Design Patterns* 一书中的模式更像配方。

因为本书特别专注于 Java，大多数模式描述与 *Design Patterns* 一书中的相比，完整性和广度较差。例如，关于引用现存系统中更大范例的已知使用(known uses)讨论很少。虽然这些模式可在各种已存在的 Java 程序中见到，但该语言仍然太新，以致不会有许多重要的用法范例。同时，一些范例模式集中于面向对象的设计，这些设计一旦完成，就可以利用其开发并发技术，这种技术以比此处容纳的更为详细的方式在一些易获取的来源中得到。

本书中的大多数技术和模式通过一些范例来举例说明，这些范例是令人困扰的小玩具运行范例集的变体。这并不是一种要令人厌烦的努力，而是为了更为清晰。并发构造常常太微妙，以至于在某些有意义的范例中迷失。小范例的重用使得模式间的小而关键的不同更为显著。同时，介绍内容还包括许多说明 Java 实现技术的代码概要和片段。

大多数并发 Java 应用程序只使用少数几种在本书中所介绍的设计，以适于手边问题为准。所有这些设计可在 Java 1.0 中实现。许多范例依赖于标准 Java 包中的类，包括 java.awt 和 java.applet，但不依赖于这些包中的特殊并发属性，不需要其它特殊工具或扩展。

1.2.1 各章预览

每章对概念原理进行了一般讨论并介绍特殊设计形式的模式风格。每章以列出相关书籍和文章的深入阅读小节结束，有时正文中省略了较不重要的方面。

本章的其余部分介绍了 Java 并发构造的引导性范例，以及对它们属性的引用概述。深入阅读小节包括一个广泛用途的资料来源主列表，这些资料是关于并发性和 OO 开发问题的。

第 2 章讨论多线程上下文中的安全性(safety)中心概念，并介绍了三种导致安全设计的保守策略(基于不变性、同步和包含)。

第 3 章讨论同等中心的概念活性(liveness)，并介绍了两种可以避免或减少活性和效率问题的一般用途的技术(分析实例变量和划分同步)。

第 4 章处理动作依赖于状态(state-dependent)(即除非对象处于适当的状态，否则不能保持成功)时适用的设计。该章描述基于状态的技术，这些技术用来避免和延迟的动作以及从不想要的效果中恢复的技术。

第 5 章介绍了对并发性控制(concurrency control)，同步分层和对基本功能的控制的三种途径。子类化、授权和元-级控制提供了以自底向上方式组合并发部件的基础。

第 6 章为创建并调用线程来执行服务(services)设计了选项。它介绍一组可根据特定上下文和应用程序修整的设计选项，实现技术和模式。

第 7 章描述流体系结构(flow architerture)——应用程序级模式，通过将充当生产者和消费者的对象间通信模式标准化，使多线程活动结构化。

第 8 章考察三种协调协同交互作用，多线程间的独立对象的途径：事务处理(transaction)、通知(notification)和调度(scheduling)技术，提供了用来克服协同设计内复杂性的工具。

可以通过 World Wide Web 链接到 http://java.sun.com/Series，来访问联机补充帮助。

该补充帮助包括：

- 全部源代码，包括只在本书中简要概述的，以及那些作为 applets 实现的范例的完整版本。
- 范例应用程序。为弥补范例中缺乏的变化，补充帮助包括了几个应用程序，介绍使用模式建立程序时面临的问题。
- 链接到包含相关信息的站点。本书中的参考列出了易于访问的书籍和文章。补充帮助中包含可电子化获取的论文、报告和网页。
- 有关本书描述模式的超链接概要，介绍了它们的附属性和关系。本书中的概念和讨论被顺序地组织。但每章中介绍的模式可同来自不同章节的其它模式一同使用。超链接化的概要使得一旦知道它们为何存在时，可以更容易地将模式付之实践。
- 其它扩展、补充、更正、奉献的范例和在本书出版后的评论，包括在 Java 1.0 版引入的新的 Java 特性用法的讨论中。

1.3 Java 并发性支持

Java 只包含少数专门为支持并发程序设计而设计的基本构造和类：

- 类 java.lang.Thread(和少数相关的实用工具类)，用于初始化和控制新的活动。
- 关键字 synchronized 和 volatile，用于控制可能参与多线程的对象中代码的执行。
- 方法 wait，notify，和 notifyAll，在 java.lang.Object 中定义。用于协调线程间的活动。

以少数支持顺序程序设计(主要为调用和返回)的构造函数导致广泛的编码实践、惯用法和设计策略同样的方式，一些并发构造在开辟新的程序设计远景上有重要意义。本节介绍这些构造的范例，以及框起来的对记号和约定的概述。它以对主要的 Java 并发构造和它们的属性的引用概述作为结束。

但首先，是少许术语：OO 程序中的交互作用需要考虑一个待执行动作的客户(client)对象和包含执行动作代码的服务器(server)对象所承担的职责。术语客户(client)和服务器(server)在此用的是其一般的含义，而不是分布式客户/服务器体系结构中的特定含义。客户只是发送请求给另一个 Java 对象的任何 Java 对象，而服务器只是接收这种请求的任何对象。

大多数对象同时扮演客户和服务器的角色。当讨论的对象是充当客户还是服务器，或两者兼有这个问题并不重要时，它被作为主机(host)引用，其它可能同其有交互作用的对象常称为协助者(helpers)或对等体(peers)。此外，当讨论 Java 的 obj.msg(arg)形式的调用机制时，接收者(即绑定到变量 obj 的对象)称为目标(target)对象。

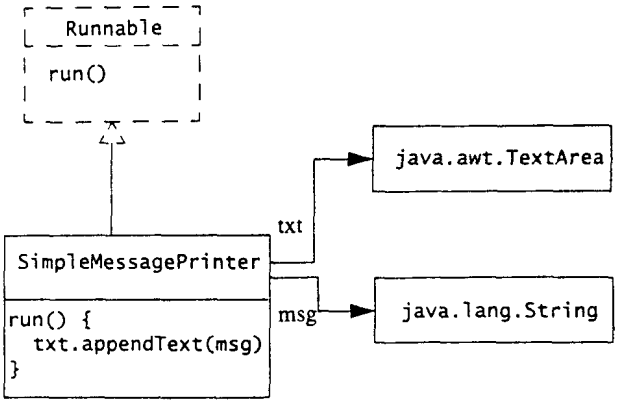
1.3.1 线程

在标准的客户——服务器交互作用的顺序形式中，客户调用服务器上的方法，然后在继续执行之前等待服务器的计算(和服务器所访问的所有对象的计算)。线程则提供了另一种选择。Java 的 Thread 框架允许活动在新的线程中被初始化，导致它被异步执行。几乎所有类型的活动都可在 Java 的 Thread 中执行。

1.3.1.1 *范例*

这里是一个描述对象的类，该对象在其 run 方法被调用时向 java.awt.TextArea 打印消

息。(TextArea 是可滚动文本区,在 Java applets 中可用于显示文本输出。)




```
public class SimpleMessagePrinter implements Runnable {
    protected String msg_;        // The message to print
    protected TextArea txt_;      // The place to print it

    public SimpleMessagePrinter(String m,TextArea txt) {
      msg_=m;
      txt_=txt;
    }

    public void run() {
      txt_.appendText(msg_);  // display the message
    }
}
```

类图表 类和接口图表使用在 *Design Patterns* 一书中用到的简化 OMT 记号的一种小变体:

- 实线框代表类。框有两部分:

类 名
方法 可选代码,伪代码或注解

- 当无需列出任何方法时,框的方法(methods)部分被省略。即使当它出现时,限定符、参数、返回类型、构造函数、继承的方法和未引用的方法也常常被省略以降低杂乱程度。
- 虚线框代表接口。
- 带有三角形的线代表子类化(extends)。
- 带虚线三角形的线代表接口实现(implements)。
- 带填充箭头的线代表从一个类实例到另一个的引用(通常由实例变量实现)。当不必展示时,它们也被省略。