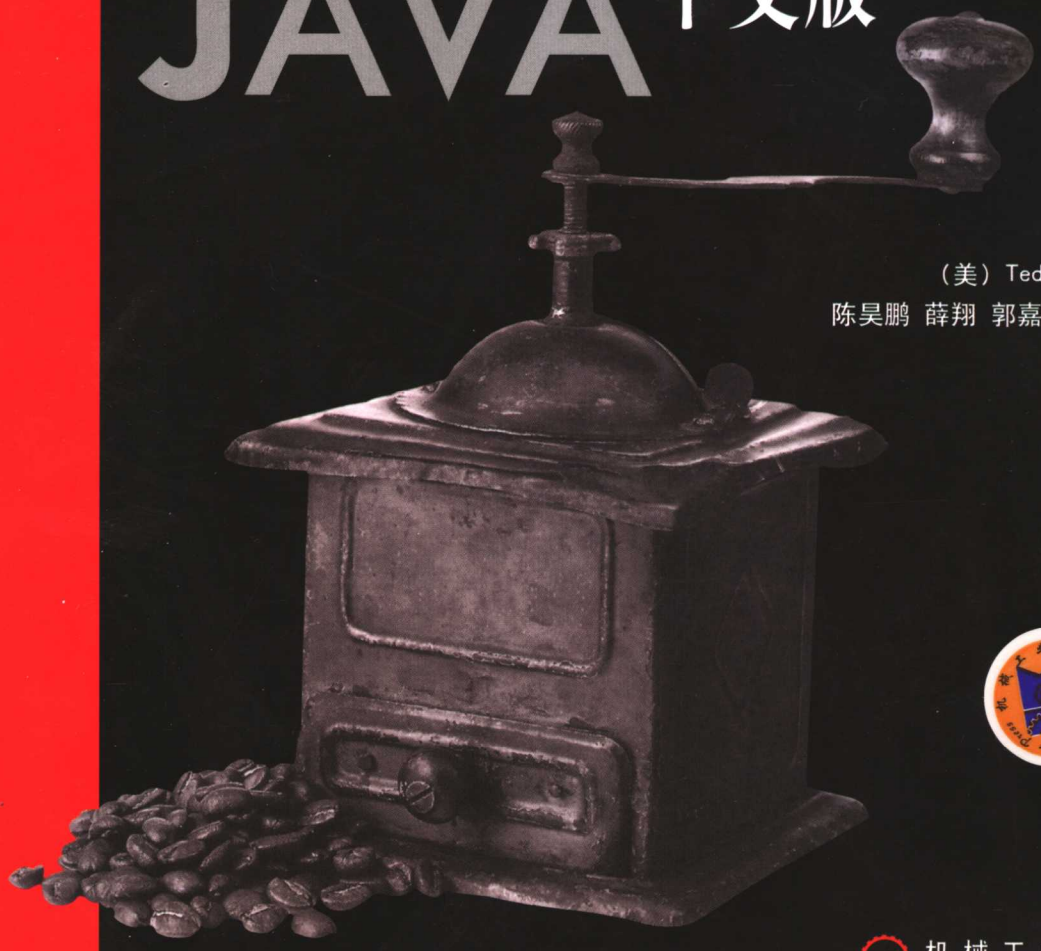
HZ BOOKS

PEARSON
Addison Wesley

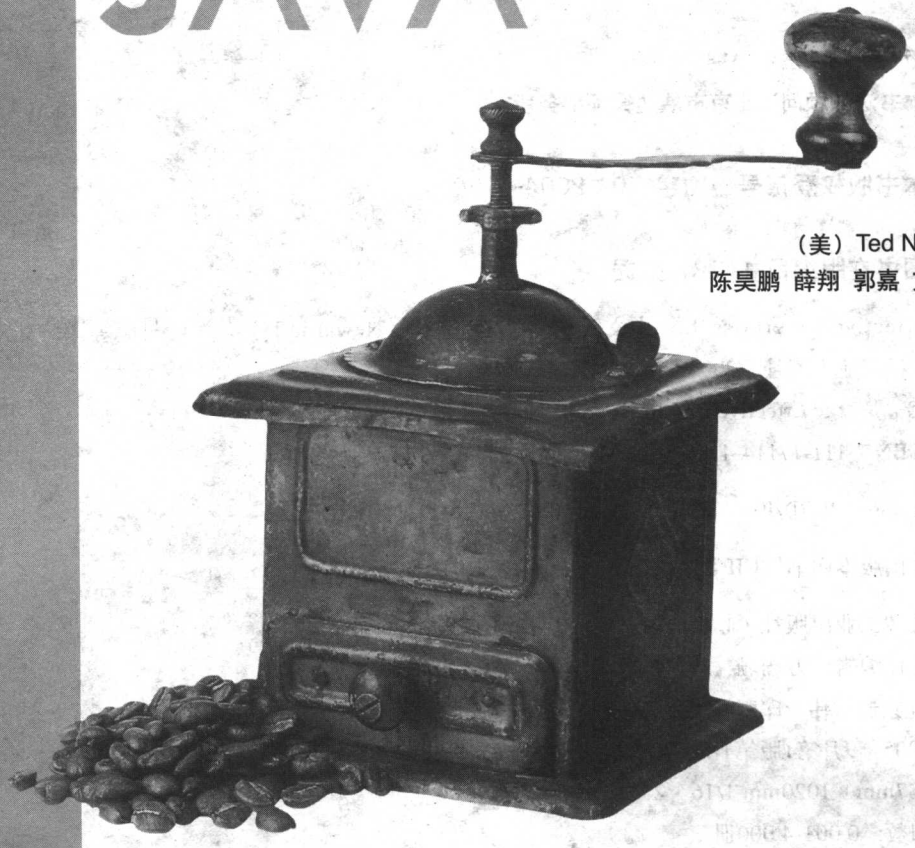
Effective Enterprise Java 中文版

(美) Ted Neward 著
陈昊鹏 薛翔 郭嘉 方小丽 译



机械工业出版社
China Machine Press

Effective ENTERPRISE JAVA 中文版



（美）Ted Neward 著
陈昊鹏 薛翔 郭嘉 方小丽 译

机械工业出版社
China Machine Press

构建高效的Java企业级系统是项困难的工作。本书详细介绍企业级计算技术中的常见问题，并描述使用企业级Java平台技术处理这些问题的方法。本书以若干条建议、指南的形式，言简意赅地介绍了J2EE开发中的微妙之处。无论你是否是Java开发人员，本书都将为你开发高效的企业系统提供诸多帮助。

本书版权登记号：图字：01-2004-6190

图书在版编目（CIP）数据

Effective Enterprise Java中文版 /（美）纽华德（Neward, T.）著；陈昊鹏等译. - 北京：机械工业出版社，2005.9

书名原文：Effective Enterprise Java

ISBN 7-111-17114-4

Ⅰ.E… Ⅱ.①纽… ②陈… Ⅲ.Java语言－程序设计 Ⅳ.TP312

中国版本图书馆CIP数据核字（2005）第091016号

机械工业出版社（北京市西城区百万庄大街22号　邮政编码　100037）

责任编辑：万珊珊　刘立卿

北京诚信伟业印刷有限公司印刷 · 新华书店北京发行所发行

2005年9月第1版第1次印刷

787mm × 1020mm 1/16 · 20.25印张

印数：0 001-4 000册

定价：45.00元

凡购本书，如有倒页、脱页、缺页，由本社发行部调换

本社购书热线：（010）68326294

序

大规模企业级系统的设计与实现难度很大，要构建高效的Java企业级系统就更加困难了。这些难题对我来说已经司空见惯。在为企业级项目做咨询的时候，我常常会遇到开发者面临的这类现实问题。在TheServerSide.com（企业级Java社区网站）上，我也经常看到针对此类问题的探讨、它们引起的困扰以及相应的解决方案。当开发者面临着J2EE这个新领域时，许多问题随之而来，TheServerSide.com正是针对开发者的需求而发展壮大起来的。它是我们的交流场所——在这里我们能够就使用的解决方案进行探讨，它也同时见证了企业级Java设计模式的发展历程。

与构建小型和单独使用的应用程序相比，企业级系统的开发是非常不同的。我们不得不去考虑那些以前确实可以忽略的问题。一旦我们要在多个用户中共享数据，就迈向了企业级系统之路。但问题也随之而来：对这些数据进行并发访问的最佳策略是什么？要在多大程度上保证数据的一致性与正确性？我们能够从2个客户扩展到50个，甚至1000个客户吗？这些都是典型问题，我觉得普通开发者在解决这些问题的时候，并不能得到足够的帮助。当然，也许我们并不应该仅仅关注于问题的答案。我们还需要学习与问题相关的各种因素，以及遇到不同问题时能够提供帮助的各种技术。有了Ted Neward的这本书，我们就拥有了这些知识，当遇到特定的问题，我们就能在解决方案中做出正确的权衡与取舍。

还没有哪本书能像这本书这样专门针对这类问题的。本书最重要的部分是，它将真正教会你两件事情。

你将理解企业级计算技术中的常规问题。

这些企业级问题并不新鲜，Ted一直专注于这个领域，并且他理解问题的核心所在。因此，即使是非Java开发者也能从本书中获益。只要你还在开发企业级系统，那么在这里学到的知识就会一直陪伴着你。语言和API也许会发生变化，但是你将会理解：构建良好架构所要考虑的问题；有哪些通信方式可供选择；如何选择状态存储的位置；各式各样的安全问题等等。

你将能够使用企业级Java平台技术来处理这些问题。

本书不仅为常规的企业级问题提供了真知灼见，而且还可以教会你采用当今的企业级Java技术来解决问题。当你把各种企业级Java技术组合在一起来考虑时，你的理解将更进一步。何时使用Web服务？消息通信能起什么作用？EJB适合做什么？本书对这些问题进行了解答。

对这些常见问题能有现成的解答，那真是太棒了。本书采用一系列高效“项”的风格来编排正好符合这一点。让我们全神贯注，体会阅读本书的乐趣吧！

Dion Almaer

TheServerSide.com主编

前　言

忘记过去的人，必将重蹈覆辙。

——George Santayana

对Java程序员来说，现在是大好时机。尽管Java作为可用的商业产品还不到10年，但在几乎所有主流计算平台上，它已经成为企业级系统的构建语言之一。那些要解决挑战性问题的公司和个人，越来越拥护Java语言及其平台。对于那些不使用Java的人来说，现在面临的问题不在于是否采用Java技术，而是准备何时开始采用。超越并包含了Java语言本身的Java 2企业级平台（J2EE）规范，涵盖了大量规范和程序库。这使得在不牺牲性能，或者不用从头实现常用算法及数据结构的情况下，也可以编写出丰富、复杂的系统。Java语言和虚拟机还在变得更加强大。针对Java开发者的工具和环境也在变得更加丰富和可靠。在许多应用领域，已经出现了大量商业程序库，这就降低了需要编写的代码数量。

大约10年以前，Scott Meyers在他的《More Effective C++》[Meyers97]一书中写下了类似的开场白。稍加改动，那一段文字就非常适合作为本书的开场白。其实，我故意模仿那一段写下了以上文字。从许多方面来看，我们会发现自己正处于Java的黄金时代，它所覆盖的领域是如此宽广，以至于Java似乎显得无所不能。正如C++在1996年居主导地位一样，Java在2004年占据了统治地位。

进行这种对比的主要目的是帮助我们认清形势。就在Scott写下那段话之后不到两年，C++就被异军突起的Java赶下了宝座。就像C++开发者开始变得“无所不能”一样，Java这门新语言及环境一出现就备受欢迎，并且几乎是一夜之间就取而代之。不过，Java现在面临着微软公司.NET平台的激烈竞争。人们自然是希望历史不要重演。要做到这一点，Java开发者就务必要使自己开发出来的系统能够达到、甚至超过原来的期望。而要达到这个目的，他们就需要知道如何才能充分利用自己所使用的语言和平台。

许多人在很多场合都曾经说过，要“领悟”一项技术，找到它的最佳使用方式，大约需要5年时间。C++正是如此：1990年的时候，我们只不过把C++看成一门新的面向对象语言，因而对它的使用也就沿袭了从Smalltalk得来的一些比较好的实践经验；到了1995年，我们就已经超越了这个程度，开始探究C++提供的独特功能（比如模板和STL）。当然，HTTP也是如此：1995年浏览器初次亮相的时候，我们把HTTP当做HTML的传输方式；如今，我们则把HTTP看成一种通用传输机制，用它来传输各种数据。

所以，从时机来说，Java是幸运的。它于1995年正式推出；但实际情况是，Java真正为普通开发者所重视是在1997年，或者说，直到那时Java才好到足以让那些批评者和怀疑者信服。近

10年以来，我们已经在大多数情况下采用Java编写应用程序，我们已经开始认识到许多实践经验和模式，并利用它们来帮助（不过还不能确保）我们进行成功的开发。作为一个群体，我们才刚刚步入正轨。

很多情形与C++时代很相似，我们又面临着10年前在C++身上出现的类似问题（Scott已经做出了回答），只不过现在换成了Java。随着语言和环境的成熟，以及我们使用Java经验的增加，我们所需要的信息也在不断变化。1996年，人们想知道什么是Java。“不管它是什么，总之和因特网有关”是一种常见的解释。刚开始，开发者着眼于使用applet，编写丰富的浏览器客户端程序，以及利用Java的跨平台移植能力。到了1998年，他们则希望知道如何使之工作：“我如何才能访问关系数据库？如何进行国际化？如何才能跨越物理机器的边界？”如今，Java程序员又提出了更高层次的问题：“应该怎样设计企业级系统，使之能适应未来的需求？如何才能提高代码的效率，而不用牺牲代码的正确性或者易用性？如何才能实现那些不被语言或平台直接支持的高级功能？”

就好像这些还不够似的，在整个企业级系统领域中，又诞生了一个新势力，它就是Web服务。尽管Java开发者还在挑战那些更高层次的Java难题，他们又面临Web服务的学习周期：什么是Web服务？它如何工作？还有（也许是最重要的），它和Java有什么关系？

在本书中，我将回答这类问题。

关于本书中的“项”

在进行深入讨论之前，我觉得有必要指出（读者也许已经注意到了）：本书与其他书籍相比，比如《Effective Java》[Bloch]和《Effective C++》[Meyers95]，书中的“项”（或者称“条”）有很大不同。特别是本书中“项”的范围要比其他同类书中的要大得多——在这里我们没有把太多的注意力放在语言或API上，而是集中于设计层的元素和使用模式上。

这其实是有意为之，我认为，对应于范围更大的企业级应用系统，这种编排方式能够在总体上与之相一致。当然，毫无疑问，所有《Effective Java》中的“项”同样可以应用于企业级应用的构建，但仅仅停留在这个层次上还不能把握住重点，企业系统有更多的内容需要考虑，而这些内容很多都超出了语言或API范围。

例如，不成功的EJB应用通常并不是源于对某个方法或接口的误用，而是由于客户直接调用的实体bean设计不佳。设计上的问题远比实现上的问题严重，要解决它，则需要在更“高层”的角度来考量实体bean大体上要提供何种功能（关于实体bean及其细节部分的讨论请参阅第40项）。

为此，本书中的“项”试图帮助开发者在系统和架构的层次，而不是在语言的层次上认识效率。许多项对某些人来说比较熟悉，还有一些只是对读者早已熟知的内容加以提炼。这样很好：对于某些读者来说是习以为常的内容，对于其他读者来说可能是全新的，反之亦然。

另外，我尽量避免讨论常见问题。已经有很多讨论最佳实践和高效使用虚拟机和语言的书了（请查看参考书目列表）。因此，本书中不会重复那些其他书中已经讨论过的内容，除非它们

在企业领域中特别适用或有某种特殊应用。

出于这个目的，我将频繁引用那些在有关企业级Java的书籍中已经建立的模式；特别是，与其他书（见参考书目）相比，我更加倾向于Fowler的《Patterns of Enterprise Application Architecture》(Addison-Wesley，2002)，Hohpe和Woolf的《Enterprise Application Integration》(Addison-Wesley，2004)，以及Alur、Crupi和Malks的《Core J2EE Patterns，2nd Edition》(Addison-Wesley，2003)。

在那些用名字来引用模式的地方，我使用标准的Gang-of-Four 模式引用格式，将页号放在引用名后面；但是，因为这些模式的来源不同，所以我也把作者的名字（比如在《Design Patterns》中用GOF来表示Gang-of-Four）作为引用的一部分。所以对Fowler的《Patterns of Enterprise Application Architecture》中的“数据传输对象”（Data Transfer Object）模式的引用就写成“Data Transfer Object（[Fowler，401]）”。

致谢

作者都愿意花很多时间来致谢；这是事出有因的。

首先，我想感谢业界的同仁，我在DevelopMentor的许多同事。Kevin Jones、Brian Maso、Stu Halloway、Simon Horrell、Dan Weston和Bob Beauchemin，他们在我编写本书的30个月中为本书的主题提供了多方面的宝贵意见。Tim Ewald、Don Box、Fritz Onion、Keith Brown、Mike Woodring、Ingo Rammer和Peter Drayton，他们通过纠正我所做的与Java无关的偏见和假设，使我深入到Java平台中。除在DevelopMentor的同事之外，我在NoFluffJustStuff研讨会上的同事，演讲人Dion Alamer、Bruce Tate、Mike Clark、Erik Hatcher、Glenn Vandenburg、Dave Thomas、Jason Hunter、James Duncan Davidson和 Ron Bodkin以及其他人，他们质疑我的结论，迫使我去证明我的断言，他们还提供了使本书更出色的建议和技巧。感谢Jay Zimmerman首先邀请我参与NoFluffJustStuff。会上很多演讲者（太多了，在此不能一一列举）或多或少都扮演了类似的角色，让我时刻自省。

第二，我要向Addison-Wesley的工作人员致敬，本书所花的时间是预计的两倍，如果没有启动这个项目的编辑Mike Hendrickson，和接手这个项目的编辑Ann Sellers，这本书永远不会完成，他们总是非常礼貌地询问本书的进展，他们的耐心远胜于我，令人印象深刻。复审人员，无论是对我放在blog上的内容，还是在后来的手稿的审查中，都做了非常出色的工作，阅读所有材料并提供了很多修正、建议。感谢Matt Anderson、Kevin Bentley、Dave Cooke、Mary Dageforde、Kevin Davis、Matthew P. Johnson和Bruce Scharlau的帮助。

写作本书既是一次热情的释放，也是一次可怕的恐怖经历。我总是在寻找这样的一个项目来表达我对企业级Java开发的想法，很少有书像以前的Effective系列一样树立榜样。《Effective C++》中，Scott Meyers为我（和其他上百万新生的C++程序员）提供了开始使用C++所需的帮助，而不仅仅是旁敲侧击。然后Joshua Bloch 写了《Effective Java》，把漂亮的“项”格式引入了Java平台。Elliote Rusty Harold进一步在《Effective XML》中沿袭了这种风格。如果有作者正

在寻找一些可追随的榜样，以上三位就是最好的人选。很幸运，我从Scott Meyers处得到了帮助，他花了几乎和我写书一样的时间来复审本书并提出建议（以积极的口吻）。他的评论和见解帮助我把一些很好的思路融入到现在你面前的这本书中。Scott，我欠你一个很大的人情，既是为了过去一年中你对我的帮助，也因为10年前当我挣扎于理解C++时你给我的指导。能和你一起工作是一种特权，同时也是我的荣幸，谢谢你。

最后，当然，我必须感谢我的家人和朋友，那些时常出现的干扰和把我拖出去玩的可爱的人们，他们的追打和尖叫时常“打扰”我，把我带回到他们称之为现实世界的地方：聚会，度假，甚至连续一两个晚上玩Nintendo64或Xbox。尽管我不会大声承认，但在过去的两年中，他们让我能以健康的心态面对写作本书的压力。

报告bug，提出建议，以及得到本书的最新资料

我已经试图让本书尽可能地准确、易读、易用，不过我知道依然有提高的空间（提高的空间总是存在的）。如果你找到了任何一个错误：技术上的、语法上的、排版上的、精神上的，不管是什么，请告诉我。我会尽力保证在后续版本中纠正这些错误。如果是你首先报告了某个错误，我会很高兴把你的名字加到本书的致谢中。同样，如果你对提高本书的后续版本质量有任何建议或想法，我也洗耳恭听。

我将继续收集那些高效的Java企业编程指南。如果你对新的指南有什么想法，并愿意与我分享，我将感到非常荣幸。你可以通过一些公开的Java 编程邮件列表找到我，最好是DISCUSS.DEVELOP.COM上的ADVANCED-JAVA列表，或者你可以通过下面的地址与我联系：

Ted Neward

c/o Addison-Wesley Professional/Prentice Hall PTR

Pearson Technology Group

75 Arlington St., Suite 300

Boston, MA 02116

你也可以给我的邮箱ted@neward.net发邮件。

从第一次印刷起，我就在本书的blog上（http://www.neward.net/ted/EEJ/index.jsp）维护着本书的修改列表（包括修订、澄清、注释和技术上的更新）。要是你愿意与其他读者分享，请把意见和勘误表贴上去。

差不多了，好戏上演了！

缩 略 语

ACID	原子性，一致性，隔离性，持久性	atomic, consistent, isolated, and durable
AWT	抽象窗口工具包	Abstract Windowing Toolkit
BLOB	二进制大型对象	Binary Large Object
BMP	bean管理的持久性	Bean-Managed Persistence
CMP	容器管理的持久性	Container-Managed Persistence
COM	构件对象模型	Component Object Model
CORBA	公共对象请求代理体系结构	Common Object Request Broker Architecture
CSS	层叠样式表	Cascading Style Sheet
DCOM	分布式COM	Distributed COM
DHTML	动态HTML	Dynamic HTML
DMZ	非军事区	demilitarized zone
DNS	域名系统	Domain Name System
DOM	文档对象模型	Document Object Model
DTC	分布式事务控制器	distributed transaction controller
EJB	企业级Java Bean	Enterprise Java Bean
HTML	超文本标记语言	Hyper-Text Markup Language
HTTP	超文本传输协议	Hyper-Text Transmission Protocol
IDL	接口描述语言	Interface Description Language
IIOP	基于因特网的ORB间协议	Internet Inter-Orb Protocol
IPC	进程间通信	interprocess communication
J2EE	Java 2企业版	Java 2 Enterprise Edition
J2SE	Java 2标准版	Java 2 Standard Edition
JAAS	Java认证与授权服务	Java Authentication and Authorization Service
JAXB	Java XML绑定API	Java API for XML Binding
JAXM	Java XML消息API	Java API for XML Messaging
JAXP	Java XML解析 API	Java API for XML Parsing
JAX-RPC	Java XML远程过程调用API	Java API for XML RPC
JCA	Java连接器API	Java Connector API
JDBC	Java数据库连接	Java DataBase Connectivity
JDK	Java开发工具包	Java Development Kit

JDO	Java数据对象	Java Data Object
JESS	Java专家系统shell	Java Expert System Shell
JIT	即时	just-in-time
JITA	即时激活	just-in-time activation
JMS	Java消息服务	Java Message Service
JMX	Java管理扩展	Java Management Extensions
JNDI	Java命名和目录接口	Java Naming and Directory Interface
JNI	Java本地接口	Java Native Interface
JNLP	Java网络启动协议	Java Network Launch Protocol
JRE	Java运行时环境	Java Runtime Environment
JSP	Java服务端页面	Java Server Pages
JSR	Java规范提案	Java Specification Request
JSSE	Java安全套接字扩展	Java Secure Sockets Extension
JTA	Java事务API	Java Transaction API
JVM	Java虚拟机	Java Virtual Machine
JVMDI	JVM调试接口	Java Virtual Machine Debug Interface
JVMPI	JVM性能测量接口	Java Virtual Machine Profiler Interface
JVMTI	JVM工具接口	Java Virtual Machine Tools Interface
LAN	局域网	local area network
MDB	消息驱动bean	Message-Driven Beans
MIB	消息信息块	Message Information Block
MVC	模型－视图－控制器	Model-View-Controller
NAT	网络地址转换	Network Address Translation
NFS	网络文件系统	Network File System
OODBMS	面向对象数据库管理系统	object-oriented database management system
ORB	对象请求代理	Object Request Broker
OSI	开放系统互联	Open System Interconnection
OWASP	开放Web应用安全项目	Open Web Application Security Project
POJO	普通Java对象	plain old Java objects
POP3	邮局协议3	Post Office Protocol v3
RDBMS	关系型数据库管理系统	relational database management system
RMI	远程方法调用	Remote Method Invocation
RMI/IIOP	采用IIOP的RMI	RMI over IIOP
RMI/JRMP	采用JRMP的RMI	RMI over Java Remote Method Protocol
RPC	远程过程调用	Remote Procedure Call

SAX	XML流解析API	Streaming API for XML
SMTP	简单邮件传输协议	Simple Mail Transport Protocol
SNMP	简单网络管理协议	Simple Network Management Protocol
SOAP	简单对象访问协议	Simple Object Access Protocol
SSL	安全套接字层	Secure Sockets Layer
STL	标准模板库	Standard Template Library
SWT	标准窗口部件工具包	Standard Widget Toolkit
TLS	传输层安全性	Transport Layer Security
TPC	两阶段提交	two-phase commit
TTL	生存周期时间值	time-to-live value
URI	统一资源标识符	Universal Resource Identifier
URL	统一资源定位符	Universal Resource Locator
URN	统一资源名称	Universal Resource Name
VM	虚拟机	virtual machine
W3C	万维网联盟	World Wide Web Consortium
WSDL	Web服务定义语言	Web Services Definition Language
WS-I	Web服务协同工作能力	Web Services-Interoperability
XML	扩展标记语言	Extensible Markup Language
XSD	XML模型定义	XML Schema Definition
XSLT	“XSL：传输，一般也写成XSL:T”	XSL:Transformation, commonly also written as XSL:T

目　录

第1章 简 介

在我来这儿之前，我对这个主题非常困惑。当我听完您的讲座之后，我仍然很困惑，但这是在更高层次上的困惑。

——Enrico Fermi

这本书展示了如何去设计和实现更加有效的企业级规模的Java软件系统：它们更有可能正确地运行，面对异常时更加健壮，更加有效率，更加高性能，更加可扩展，更不易被错误地使用。一句话，这些软件就是更优良。

然而，为了做到这一点，我需要明确指出本书应该覆盖的内容与不应该覆盖的内容。特别是，本书不是在对如何使用语言本身的那些有效的技巧进行改头换面的重新包装——那些内容属于Joshua Bloch的优秀著作《Effective Java》[Bloch]所讨论的领域，那本书应该被看作是所有Java程序员的必备读物。而本书则面向更高的层次，即为企业系统编写Java软件的各种技巧，因此，本书具名为《Effective Enterprise Java》。

这样，至少对我们的目标来说，精确地定义什么是一个“企业Java”系统显得非常重要。对许多开发者来说，在这里，有关关系型数据库、业务规则、事务功能以及可扩展性的讨论是处于重要地位的。任何运用了在J2EE规范中被定义的大多数规范的系统自然都应是讨论对象。然而，我却倾向于从另一个稍显不同的角度来考虑问题的答案。

一个企业系统是具备下列性质的系统：

- *共享某些或全部在应用中被使用的资源*：这里普遍存在的例子就是所有的应用数据驻留的关系型数据库。共享这些资源会增加额外的隐含复杂性：数据被共享是因为它需要同时对多个用户可用。因此，系统必须支持安全且快捷的并发用户访问。
- *规划成为内部使用*：这里的“内部”指的是“大量生产的卖给最终用户的软件的对立物”。当系统确实可以在公司与商业伙伴之间共享时，它可以用公司的特有知识、商业惯例和特殊需求来编写。
- *必须在现有的架构内运行*：在极特殊的情况下，公司才有可能已经拥有了一套适当的系统必须能够与之进行互操作的硬件和软件。特别是，这意味着应用必须适应现有的数据库模式（而不是其他的方式）。一个企业系统必须能够适应它所存活的异构系统。
- *将由内部IT员工部署并提供支持*：对大多数公司来说，实际的“产品”生产都超出了开发者的职责范围。这是一件好事情——大多数开发者都会对因为他们开发应用出现的故障，在凌晨时分被唤醒而感到不痛快。但是这同时也意味着系统的部署将要由他们之外的人手去完成，并且这还意味着负责数据中心的员工必须有某种办法在未经历编写代码环节的情

况下，去监视、诊断和订正问题。

- 需要更强的健壮性，对于异常处理和可扩展性都是如此：企业系统，特别是通过因特网可访问的系统（马上就能想到经典的e-commerce系统），象征着某个公司的一项巨额投资。系统宕机的每一分钟都意味着成千上万，也可能是成百万上千万美元收入的流失。每一个厌恶公司网站的用户（更糟的是，强制用户紧盯浏览器去等待登录请求的完成）都会导致公司的诚信度和潜在销售额的损失。
- 只能适度地失败：在像字处理器这样的应用中，某个意外情形可以通过抛出一个“唉呀”对话框，保存用户正在进行的工作，并请求他或她重新启动程序来处理。一个企业系统却不能这么做——如果它崩溃了，成百万上千万美元就会因丧失了生产率、销售额、客户感知等等而付之东流。一个企业系统要为“五个九的可用性”而奋斗：系统每年全部的正常运行时间要大于99.999%。这使得系统的停工期，不论是预先安排的还是突发的，只剩下每年0.001%，或者粗略计时为少于5分钟。
- 必须合理地处理系统随时间推移而发生的演化：企业系统具有长久的生命周期，Y2K（2000年）问题证明了这一点。因此，一个系统必须能够适应在公司内随时间推移而发生的不可避免的变化：合并、销售额增长、政策调整、法人变更、征购等等。

很明显，这是一个相当大的领域。企业软件在尺寸和范围上都是横跨整个领域的，从个人的电子表格软件到数个TB的关系型数据库。随着无线设备在大型公司内的使用的迅猛增长，你甚至还会主张为PalmOS设备或移动电话编写代码也属于企业开发。然而，本书在很大程度上聚焦在企业计算的传统领域内，即PC连接到一台或多台服务器上。

虽然上面的定义覆盖了一个相当大的范围。企业应用可以被内部或外部使用，可以运行完整的关键内容：某些可以在本质上纯粹是管理性的，例如人力资源部的休假报告系统，某些可以是公司的核心收入流，例如像Amazon.com这样的在线零售系统。商业伙伴可以使用企业系统来下订单、托运出货或者递交发票。咨询公司可以为顾客放置供其读取的技术信息。

所以，编写满足这些需求的软件可能会很艰难，并且很费时间。由此，J2EE应运而生。

J2EE的目标

按照老话的说法，知道我们在哪里（以及为什么我们在那里），将有助于我们去了解下一步该如何走。我想解释的是J2EE的“为什么”和“如何”，以确保对某些概念（例如就像“查找”，对第16项来说很重要）的清晰认识。

纵观计算科学的历史，任何语言、工具或库的终极目标主要都是要提高抽象级别，使我们从那些会让我们无法专注于手头实际工作的细节中抽身而出。例如，思考一下经典的OSI七层网络协议栈。当你要“打开一个套接字（socket）”时，并不是要实际打开什么能够直接连接到其他机器的东西，这个动作实际上是在软件（以及硬件，只要你涉及物理层）的四个层次或五个层次之上的抽象，其中每一层都提供了确定数量的支持，以使这些资源运转起来。

在企业系统的早期发展阶段，分层的缺乏到了令人头痛的程度——所有数据访问都是直接通

过由定长记录组成的文件而完成的，在那些记录上发生的任何事都属于你的业务，并且属于你独有的业务。之所以呈现出没有分层机制的局面，是因为在那些日子里我们运行的系统数量相对于CPU周期或内存空间来说并不算多。每件事物都必须尽其所能地使其受到关注。

随着硬件容量的增长和对复杂处理的需求的增长，我们发现必须而且很渴望使某些行为得到保障。所以一个新的软件层被置于了传统的平面文件集合之上，我们称之为事务处理系统。它管理对数据的并发访问，确保数据遵从我们通过自己编写的代码而对其施加的逻辑约束。随着时间的推移，这层软件通过引入了一个强大的查询语法而被更加形式化了，而且由此诞生了现代关系型数据库和SQL。

此后，我们开始希望让终端用户使用存储在数据库中数据，而不是给数据处理员成沓的包含了要输入数据的纸张。这绝不仅仅只是意味着大学生们丢掉了在暑期勤工俭学的一条出路，更重要的是客户/服务器架构由此而诞生。在客户机上执行的程序，要负责数据的获取和呈现，并将这些操作转换成工作语句以执行到数据库系统的操作。通常情况，这类程序都具有各种不同的图形化用户接口，它们是用为此而专门构建的某种高级语言编写的，并需要根据公司正在开发的特定系统进行定制。

然而，随着这些客户/服务器系统的客户数量的增长，我们开始陷入了局限：由于有那些伴随客户动作的内部处理，到数据库的物理网络连接（以及相关的软件开销）就有了一个确定的上限，这样，就能够给可以同时使用系统的用户数扣上一个任意大的帽子了。我们可以说n个客户端是系统用户数的上限，其中n是最大连接数，只要第n+1个客户想要登入系统，我们就需要为他或她创建新的数据库。

即使是在《财富》杂志的排行榜上最大的50家公司，在短期内也可以接受这一状况，因为一个企业系统的最大用户数通常不会超过四位数。不管在这么做时，花费了多大的代价，通常还是有可能，会将一个新的安装推给上千个内部客户端，尽管这并不是人们希望的，然而，我们一开始采用Web作为企业系统的公共界面，情况就突然间从根本上发生了变化——Web所涉及的东西都是在扩展公司“疆域”，事实上可以说，这意味着用户可以从世界上任意的地方访问公司。过去我们拥有上千个客户端的地方，现在使用Web就意味着拥有了上百万个客户端。

可能的并发客户端数量的几何级数般的跃升意味着旧有的“一个客户端，一个连接”的架构已经再也不可能了。因此急需新一代的软件架构，它能够使通过Web将系统带给最终用户的思想具有可以运作实现的机会。

计算机科学有一句经典格言叙述道：“没有任何问题是不能通过增加额外的附加层而得到解决的。”在这里，因为客户端程序通常并非在所有的时间内都在使用它们所持有的到服务器的连接，所以被引入的间接层是位于客户端和服务器之间的软件层。（注意这个精细的术语定义，详情可参阅第3项。）这个资源管理软件层，在为其起一个好名字而争论了数年之后，有了一个众所周知的名称：中间件（middleware）。

中间件和J2EE

Bernstein（在[Gray/Reuter]中被引用）将术语中间件定义为一项分布式系统服务，它包含标

准的编程接口和协议。他进一步声明道，中间件服务在操作系统和网络层之上，在具体的工业应用之下，提供了一个支持层。中间层，像以前出现的人们熟知的事务处理监视器（Transaction Processing Monitor，缩写为TP Monitor），是“为了要集成其他的系统构件和管理资源……它与许多不同的软件构件之间都有接口，它的主要目的是要以某种特殊的方式使这些软件构件协同工作，这种方式最终形成了著名的面向事务的处理（Transaction-oriented Processing）”[Gray/Reuter，240，特别强调]。

J2EE很明显是事务处理监视器/中间件遗产的一个继承者。J2EE规范自身就是将一打的其他Java规范糅合到一起组成的一个一致的、可定义的整体，它并没有描述多少新增的或附加的内容到这些规范中去，以此为利用所有这些规范来构建的系统提供一个稳定的和内部一致的基础。Servlets、JavaServerPages、JDBC、RMI（Remote Method Invocation，远程方法调用）、JMS（Java Message Service，Java消息服务）、JTA（Java Transaction API，Java事务API）、JCA（Java Connector API，Java连接器API），当然还有EJB——这些规范以及其他规范都被集合起来，形成了在J2EE规范的遮护之下的一个极为和谐的事物。

即使不是全部，那至少也是被J2EE统成一体的规范都是用来处理资源管理的。例如，JDBC描述了如何与关系型数据库系统进行交互并进行操作；JMS覆盖了集成面向消息的中间件（这里又使用了这个词）系统的内容；RMI是有关远程过程调用的；JTA是有关事务管理的；JCA是有关“遗留”（legacy）系统、其他的通信系统以及面向记录的数据系统的；还有其他等等。

有关中间件的思想在许多方面都来源于对集成的渴望。非常频繁的情况是，一个企业发现它自身拥有大量的未互联的“烟囱”系统，这些系统都被用来完成某项特殊目的的任务。（烟囱系统（stovepipe system）这个名字来源于应用面很窄、专注于某一点的应用系统的思想：一个数据库被一个单一程序访问，当我们用图形法观察这个程序时，它看起来就像是一根烟囱。）这些系统通常都是在一个单独的科室或部门的控制下而被开发出来的，它们处理该特殊科室或部门的独特的要求。这里列出部分这样的系统，包括财会、存货管理、人力资源、客户关系管理、订单登记等等。单独地看这些系统，通常每一个都是十分成功的——它们满足某个部门的需求，正是这些需求驱使该部门渴望首先构建这些系统。

遗憾的是，企业整体的需求并不止于各个部门自己的简单需求。一个次要的、中间的需求集贯穿整个企业，它们几乎是在“因特网革命”的冲击出现之时，就得到了实现。当公司开始审视使用因特网作为一种触及客户的新方式时，一旁便是它们传统的实体零售店的渠道，它们所开发的企业系统随时间推移突然变得很不适合了。公司希望将每一件事情都推到线上：订单登记、订单跟踪、供应链管理等等。他们很快发现，为那些通过电话登记订单的销售商而建立的系统，在置于HTML表格之后时就不能运行了——当订单登记突然间成为了顾客的职责时，例如，比起由训练有素的公司职员来实现这一任务，这时需要更多的验证。新的渠道也在不断地被设计和探索，现在，公司希望通过因特网把它们的货物和服务不仅兜售给顾客，同时也兜售给供应商和商业伙伴（即通常讲的“B2B”渠道）。移动设备正在迅速地吸引着人们的兴趣，这使得让商业企业通过无线PDA、蜂窝电话和其他传统的HTML不能涉及的工具向客户销售成为