摘 要

本论文的研究目的是希望通过对 J2EE 技术体系的深入研究，总结出在开发 J2EE 应用的时候容易出现的性能问题，并结合实例提出相应的解决方案。这些问题和解决方案都具有普遍的适应性，因此可供开发一般的 J2EE 项目所借鉴。

J2EE 是由 SUN 公司推出的一种全新概念的模型，目前已成为企业级应用，电子商务交易的主要开发平台。J2EE 已经被证明是一个稳定的、可扩展的、成熟的平台，它具有平台独立的特性。但是做为 J2EE 技术的核心之一的 EJB，其性能问题一直受到广泛的质疑，普遍被认为执行效率低下。正是由于 J2EE 在性能上的缺陷，因此我们在开发 J2EE 应用时，必须重视应用的优化工作，努力做到即可以充分利用 J2EE 的种种优势，又可以使项目高效的运行。

论文首先从 JAVA 语言的内存管理出发，深入研究了 JAVA 语言内存分配及回收的机制，重点研究了 JAVA 开发中可能会出现的内存泄露问题。并提出了避免内存泄露和如何检测内存泄露的方法。其次，本文深入研究了 EJB 技术的原理，并结合 JBOSS 平台研究如何提高 EJB 性能的问题。第三，本文对 J2EE 集群技术做了深入的分析，并对流行的各种集群方式的性能加以比较，提出了一种比较理想的集群方式。第四，本文对现有的多种 WEB 层负载均衡方式加以研究，发现它们都不能真实的按照各服务器结点的负载均衡情况均匀的分配请求。为此，笔者提出基于负载检测的负载均衡方案，其核心是给决定负载大小的各因素分配一定的权重，在每次请求转发时，各服务器结点按照事先分配的权重对其实际负载的大小予以量化，并将量化后的结果反馈到负载均衡器，由负载均衡器挑选其中负载最轻的服务器转发请求。第五，本文最后将前面所研究的种种优化技术综合运用到笔者目前所开发的项目中去，并取得了一定的效果。

关键词：J2EE　JBOSS　优化

ABSTRACT

The objects of this paper is to conclude some methods to increase the peformance of J2EE application at the basis of researching the Architecher of J2EE platform.

J2EE specification makes the n—tier enterprise application development much more easier, and the servers based on it offer the running environment and system serservices such as transaction,database accessing, JMS and security service. The major advantage of J2EE is its "write once, run anywhere" Java components, which means that it is independent of the operation system, and the J2EE productions can be deployed to any J2EE servers including the server in NT system environment.because the J2EE technology has so many merits.so the technology of J2EE platform gradually become the fact standard of enterprise component deployment.but,as the core of J2EE platform,EJB technology is thinked as a low efficient technology.because ejb technology are developed at the basis of Serialization technology and remote method invocation technology.Serialization is used to transform objects between processes.but the implemention of serialization is not designed peferctly in java platform,so the efficiency of serialization is not so good.the same as serialization technology,the Remote method invocation technology also faced the criticism of bad perfomance.because of the these disadvantage,developers of j2ee application must carefully develop their work to improve the perfermence of J2EE application.,

In my paper,firstly,I researched the mechanism of java memory allocation and memory recovering.I emphased on the problem of memory leak.I analysed the reason of memory leak and how to check the memory leak and how to avoid the memory leak.secondly,I reasearched how to improve the performance of the EJB components in jboss platform.thirdly,I reasearched the theory of J2EE cluster and the implementions of all kinds of cluster in the J2ee architecture in jboss platform.finally,I propose a load balancing for HTTP/Servlet level policy and make it implemented.

Key Words: J2EE, JBOSS, PERFORMANCE TUNNING

第一章 绪论

1.1 课题的研究背景与意义

本课题主要是作者在参与了 J2EE 平台应用项目后的分析工作的一部分。本课题主要研究对 J2EE 应用进行优化的若干技术问题。

J2EE 是由 SUN 公司推出的一种全新概念的模型，是在分布式环境中的一种体系结构，它提供了一种基于组件的设计、开发、集成部署企业应用系统的方法。J2EE 平台提供了多层分布式的应用系统模型，可重用的组件、统一的安全性模型和灵活的事务控制，基于组件的 J2EE 企业应用具有平台独立性，所以不受任何软件产品和任何软件厂家 API 的限制[①]。J2EE 应用已经成为企业级应用，电子商务交易的主要开发平台。符合 J2EE 应用标准的产品主要有 BEA 公司的 WEBLOGIC、IBM 的 WEBSPHERE 以及开源的 JBOSS 等，还有包括著名数据库厂商 ORACLE、SYBASE 在内的上百种产品。它们基本上都是应用服务器系统，因此 J2EE 通常指的是应用服务器平台。J2EE 已经被证明是一个稳定的、可扩展的、成熟的平台，它具有平台独立的特性。

J2EE 平台独立性包括两个方面，一是 JAVA 语言的平台独立性，二是 J2EE 标准的平台独立性. JAVA 是一种跨平台的语言，在任何平台上，只要有 JAVA VIRTUAL MACHINE（JVM），就能在不同平台上执行同一个 JAVA 程序。另外，J2EE 标准的平台独立性使得任何符合 J2EE 标准的应用服务器之间可以共用标准的组件开发。也就是说，在 WEBLOGIC 下开发的应用，也可以移植到 JBOSS 平台下，这样在电子商务应用的开发中，可以任意的选择和购买通用的组件，从而加速开发的进程。最后，J2EE 的平台独立性是应用的移植变得轻松起来。在一个 J2EE 应用中，你可以把 WEBLOGIC＋ORACLE 的应用移植到 JBOSS＋SYBASE 上，不需要改动任何源代码。只需要进行一些不算太复杂的配置即可。这也是 J2EE 为何如此流行的一个重要原因。

J2EE 技术虽然受到了广泛的欢迎，但是做为 J2EE 技术的核心之一的 EJB，其性能问题一直受到广泛的质疑，主要的质疑集中在两个方面。一是对象序列化技术，对象序列化技术是 EJB 跨平台通迅的基础，所有的 EJB 通讯都依赖于对象序列化技术，通过对序列化技术实现了对象在多个进程之间的复制传递。这本是一个很清晰的设计，但是由于 JAVA 平台对于对象序列化的实现并不尽如人意，序列化与反序列化的效率太低，导致对象传输的速度极慢。二是 RMI（远程方法调用），EJB 限定必须遵守 CORBA 规范的 RMI-IIOP 技术，这种技术的原理是通过本地的一个远程对象代理，通过网络上的多次通讯实现对远程对象的方法调

①谢小乐，《J2EE经典实例详解》，人民邮电出版社，2002，235-239

用，这种设计架构的初衷是隐藏对象的具体位置，可以让对象使用者不用关心对象的实际位置。但是这种方法的实现性能极差。由于 J2EE 技术先天上存在这些性能上缺陷，因此我们在开发 J2EE 应用时，必须重视应用的优化工作，努力做到即可以充分利用 J2EE 的种种优势，又可以使项目高效的运行。

1.2 企业计算平台的发展概况

1.2.1 传统的应用体系结构

传统的应用体系结构主要有两种 C/S(client/server)模式和 B/S 模式。

（1） C/S 模式(客户机/服务器)模式

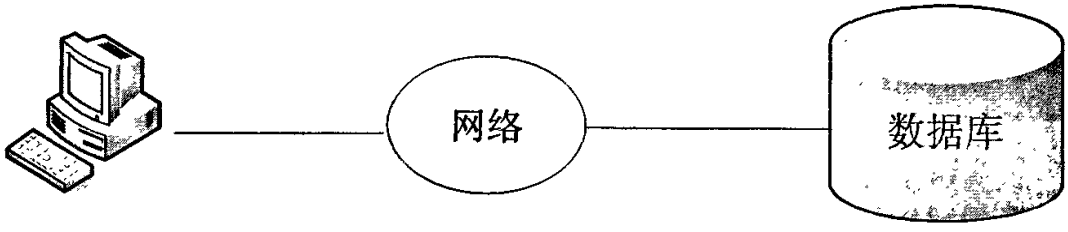

图 1.1 C/S 模式

资料来源：本研究整理

目前绝大多数网络应用都采用这种模式，它将访问数据库的操作放在客户端执行，即客户应用程序直接对数据库进行操作。一般企业的 MIS 系统都采用这种方式。

（2） B/S 模式(浏览器/服务器)模式：

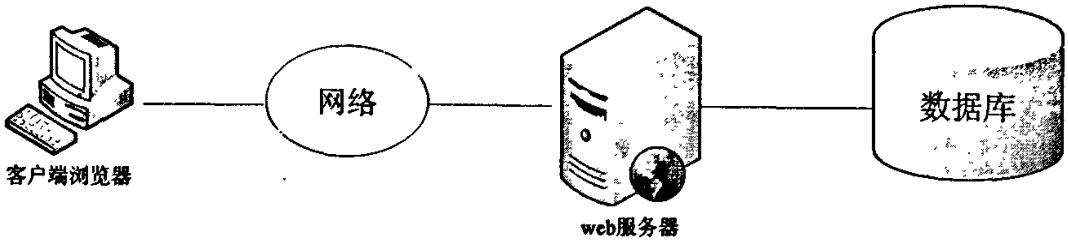

图 1.2 B/S 模式

资料来源：本研究整理

B/S 模式访问数据库的工作交由客户端的浏览器来完成。不必安装特定的程序。随着 internet/intranet 技术的发展，网络应用中的客户端零管理成了众多企业的需要，所以这种体系结构正迅速的发展起来。

1.2.2 多层应用体系结构

多层应用体系结构就是传统的两层结构的客户端与服务器之间加入一层中间层，我们称之为应用层。应用程序服务器就放在该层。应用程序服务器是驻留

在服务器上的程序，为各种应用提供了商业逻辑。服务器可以是网络的一个组成部分，尤其是分布式的网络。服务器程序为客户机上的程序提供服务。

应用程序服务器最常用的场合是基于 WEB 的三层体系结构。

第一层(前端)：BROWSER 浏览器(THIN CLIENT 瘦客户机)，为用户提供 GUI 图形接口。

第二层(中间层)：APPLICATION SERVER：应用服务器。

第三层(后端)：DATABASE SERVER,数据库服务器。

应用服务器位于三层结构中的第二层，它是三层体系的集成部分，和 WEB SERVER 联合在一起处理客户的请求。

应用服务器的基本功能包括：

（1）组件管理

提供管理工具来处理所有组件和运行时服务(RUN TIME SERVIC)如 SESSION 管理，同步/异步客户端通知，以及执行服务器上的商业逻辑。

（2）容错

提供没有任何单点错误的性能，定义出错恢复的策略以防止某个对象或对象组出现错误。

（3）负载均衡

根据服务器当前负载及可用性，将请求提交给不同的服务器。

（4）事务处理

（5）管理控制台 MANAGEMENT CONSOLE

单点图形管理界面，用于监控远端客户及服务器集群管理。

（6）安全性

提供应用的安全性。

1.3J2EE 平台的发展情况

1.3.1J2EE 的内容

J2EE 是 SUN 公司提供的旨在为支持 JAVA 语言服务器端部署而提供平台无关、可移植的、多用户的、安全和标准的企业级应用平台。J2EE 做为一个规范，包括如下几个部分：

(1)企业级 JAVABEAN® (EJB Enterprise JavaBeans)：EJB 定义了如何编写服务器端组件，并且为服务器端组件和管理这些组件的应用服务器之间提供了标准的协议。EJB 是 J2EE 中的重要组成部分，并且使用了其它的 J2EE 技术。

(2)JAVA 的远程方法调用 (RMI：Remote Method Invocation) 和 RMI－IIOP(Remote Method Invocation Over the Internet Inter-ORB Protocol)RMI

② （美）斯瑞格奈斯，《精通EJB》，电子工业出版社，2006，56-59

是 JAVA 语言自身提供的用来在分布式对象之间通信的机制，例如运行在两个不同的机器上的不同的对象之间。

(3) JAVA 命名和目录接口（JNDI, Java Naming and Directory Interface）：JNDI 用来访问命名和目录系统。

(4) JAVA 数据库连接（JDBC，Java Database Connectivity）：JDBC 是 JAVA 访问关系型数据库的 API。

(5) JAVA 事务处理 API（JTA，Java Transcation API）和 JAVA 事务处理服务（JTS，Java Transcation Service）：JTA 和 JTS 规范为组件提供了可靠的事务处理支持。

(6) JAVA 消息服务（JMS，Java Message Service）：JMS 允许 J2EE 应用通过消息进行通信。

(7) JAVA Servlet:Servlet 是一些用来扩展 WEB 服务器功能的网络组件，它基于请求/响应机制。

(8) JAVA SERVER PAGE（JSP）等。

1.3.2J2EE 应用服务器

目前流行的应用服务器如 Weblogic，Websphere Appserver，jboss, oracle 9iAs 等等，都是符合 J2EE 标准的应用服务器，很好地支持了 EJB、JNDI 等 J2EE 核心技术。

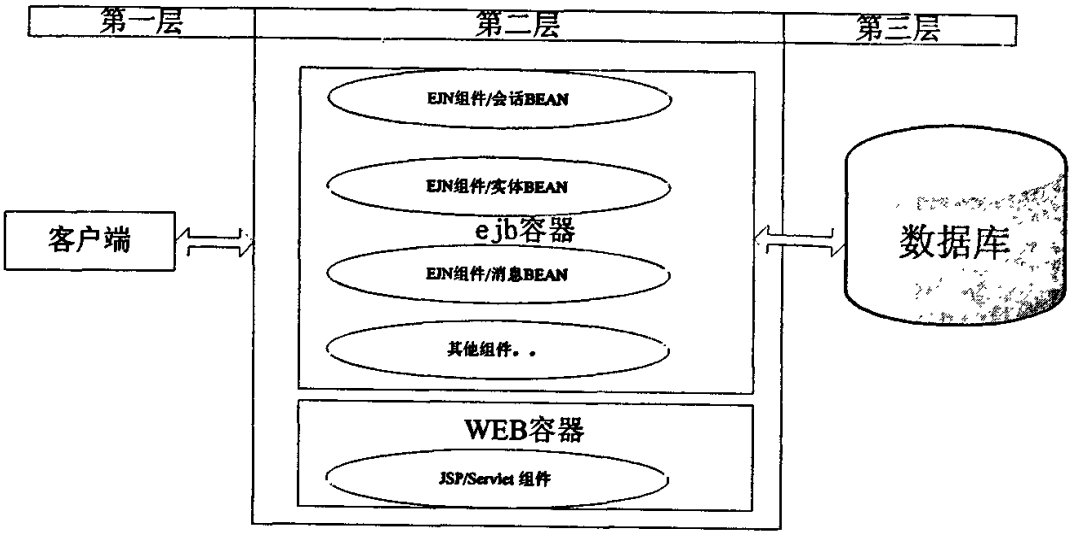

图 1.3 J2EE 组件模型

资料来源：本研究整理

J2EE 定义了下列组件：

(1) 客户层：Application

(2) Web 层：Java Servlet 和 JSP 组件

(3)业务处理层：EJB

其结构如图 1.3 所示,下面简要介绍各元素的主要功能：

(1)EJB 容器

Enterprise Java Bean 实例运行于一个 EJB 容器[3]当中。该容器是控制 EJB 并为其提供重要的系统级服务的环境。即你可以不用自己开发这些服务，而完全集中在 EJB 的业务逻辑中。该容器为 EJB 提供了以下一些服务：

a)事务管理（Transcation Management）

事务是企业级计算中的关键部分，Java Transcation API(JTA)给开发人员提供了一组简单的面向事务命令，在 JAVA 代码中可以轻松可靠地控制事务。这些事务包括 EJB 方法调用，JDBC 数据库操作、JMS 消息服务系统以及对任何其它提供事务支持的计算源操作。

b)安全（Security）

容器允许只有被授权的用户才能激活 EJB 的方法。每一个客户属于一个特别的角色，而每个角色只允许激活特定的方法，你应该在 EJB 的部署描述中声明角色和可激活的方法。由于这种声明的方法，你可以不必编写加强安全性的规则。

c)远程客户连接（Remote Client Connectivity）

容器负责管理在客户端及 EJB 之间的底层交流。EJB 被创建后，客户端可以像在一个 JVM 中一样对 EJB 激活其方法。

d)生存周期管理

一个 EJB 在其生存周期中会经历几个阶段。容器创建 EJB，并在可用实例池与活动状态中移动它，而最终将其中容器中移去。即使可以用 EJB 的 CREATE 和 REMOVE 方法，容器也会在后台自动执行。

e)数据库连接池

数据库连接池是一个有价值的资源。获取数据库连接是一项费时的工作，而且连接数据非常有限。容器通过管理连接池来缓和这些问题。EJB 可从池中获取连接，在 BEAN 释放连接后供其它连接使用。

(2)WEB 容器

WEB 容器是 JSP 和 SERVLET 的运行环境，来自客户端的请求，首先要被 WEB 容器所接收，通过 JSP 或 SERVLET 来调用 EJB 完成业务功能。

1.3.3JBOSS 应用服务器

JBoss 是一个 Java 开源、集成和开发基于 J2EE 的完整服务实现。JBoss 提供 JBossServer、基本的 EJB 容器及 JMX 框架。它也为 JMX 系统提供 JBossMQ，为 JTA 事务提供 JBossTX，为 CMP 持久化提供 JBossCMP，为基于 JAAS 的安全性

③杨绍方，<<深入掌握J2EE技术>>，科学出版社，2002,192-198

提供 JBossSX，为 JCA 提供 JBossCX。为支持 WEB 组件，比如 SERVLET 和 JSP，JBoss 提供了抽象集成层。而这些抽象层的集成服务实现可以由第三方 SERVLET 引擎提供，比如 TOMCAT 和 JETTY。JBOSS 使得开发者能够通过 JMX 混合应用这些组件，即借助于替换 JMX 兼容的组件实现完成这些任务。同时，这些 JBOSS 组件之间甚至不会产生任何影响。因此，JBOSS 现在全部都是模块化的。

1.4 本文研究内容

本文主要从以下几个方面对 J2EE 应用的优化进行研究：

(1) JAVA 应用的内存管理，提出了如何防止和检查内存泄漏问题

(2) 对 EJB 的原理进行分析，对如何通过配置 JBOSS 服务器提高 J2EE 应用的效率进行了研究。

(3) 深入研究了 J2EE 应用的各种集群技术。

(4) 提出了基于负载检测的负载均衡算法，并予以实现。

(5) 将多种优化技术应用到笔者所开发的废旧家电回收信息管理系统当中去，对优化效果予以检验。

第二章 JAVA 内存管理

内存管理话题在C或C++程序设计中讨论得相对较多，因为在C与C++程序中，需要开发人员自己申请并管理内存，开发人员可以申请/借用系统内存并且负责释放/归还系统内存，如果“只借不还”的话，就会造成系统内存泄漏的问题，在JAVA程序中，这些工作由JAVA虚拟机（JVM）负责处理。所有内存的申请、分配、释放都有由JVM负责完成。因此，开发人员就省去了这部分的工作，不过这并不意味着开发人员可以完全依赖于JVM的内存管理功能，如果开发人员在开发过程中完全不理会内存的问题，那么你所开发的应用的性能，很可能不是最优的。因为无论配置多么优良的硬件环境其自身的资源都是有限的，而内存又是硬件环境资源中重要的一部分，因此如果说如果开发人员开发的JAVA应用没能有效、合理地使用系统内存，那么就不可能具备较高的性能。下面对在JAVA应用开发中的内存管理相关的技术做一些探讨。

2.1 垃圾回收

Java的内存管理就是对象的分配和释放问题。在Java中，程序员需要通过关键字new为每个对象申请内存空间（基本类型除外），所有的对象都在堆(Heap)中分配空间。另外，对象的释放是由垃圾回收（Garbage Collection GC）决定和执行的。在Java中，内存的分配是由程序完成的，而内存的释放是由GC完成的，这种收支两条线的方法确实简化了程序员的工作。但同时，它也加重了JVM的工作。这也是Java程序运行速度较慢的原因之一。因为，GC为了能够正确释放对象，GC必须监控每一个对象的运行状态，包括对象的申请、引用、被引用、赋值等，GC都需要进行监控。监视对象状态是为了更加准确地、及时地释放对象，而释放对象的根本原则就是该对象不再被引用。

垃圾回收（Garbage Collection GC）[④]是JAVA程序设计中内存管理的核心概念，JAVA虚拟机（JVM）的内存管理机制被称为垃圾回收机制。因此，掌握如何在开发JAVA应用时合理地管理内存，首先应该了解JAVA虚拟机的内存管理机制：垃圾回收机制，不了解垃圾回收具体实现机制，JAVA程序设计中的内存管理就无从谈起。

那么在JVM运行环境中什么样的对象才是垃圾呢？下面我们给出了在JVM环境中垃圾对象的定义：

一个对象创建后放置在JVM的堆内存中（heap），当永远不再引用这个对象时，它将被JVM在堆内存(heap)中回收。被创建的对象不能再生，同时也没有办法通过程序语句释放他们。

---

④王森,<<JAVA深度历险>>,麦格罗.希尔国际出版社,2001,102-104

我们也可以这样给 JVM 中的垃圾对象下定义：

当对象在 JVM 运行空间中无法通过根集合（root set）到达时，这个对象就被称为垃圾对象。根集合是由类中的静态引用域与本地引用域组成的。

为了更好理解 GC 的工作原理，我们可以将对象考虑为有向图的顶点，将引用关系考虑为图的有向边，有向边从引用者指向被引对象。另外，每个线程对象可以作为一个图的起始顶点，例如大多程序从 main 进程开始执行，那么该图就是以 main 进程顶点开始的一棵根树。在这个有向图中，根顶点可达的对象都是有效对象，GC 将不回收这些对象。如果某个对象（连通子图）与这个根顶点不可达(注意，该图为有向图)，那么我们认为这个(这些)对象不再被引用，可以被 GC 回收。

以下，我们举一个例子说明如何用有向图表示内存管理。对于程序的每一个时刻，我们都有一个有向图表示 JVM 的内存分配情况。图 2.1 的右图，就是左边程序运行到第 6 行的示意图。

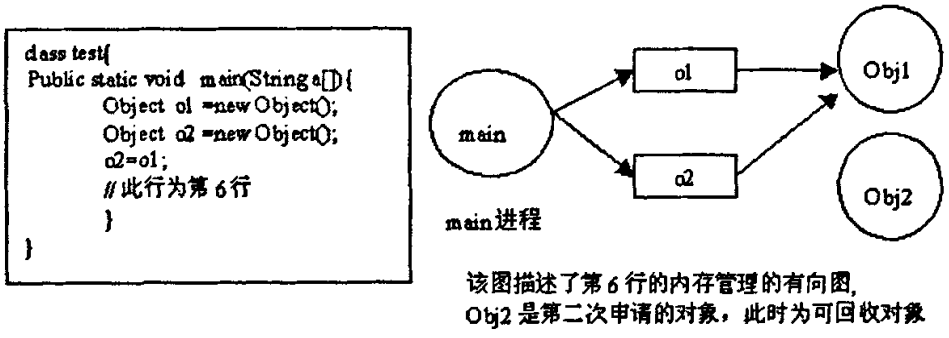

图 2.1 垃圾对象的产生

资料来源：http://www.zhujiangroad.com/html/soft/s2957.html

在这个例子中 main()函数中的 O1 与 O2 是对象的引用（指向对象的指针），构成了根集合，它们存在于给 main 函数的堆栈中，Obj1,obj2 是对象的实例，它们创建于堆中。变量 O2 最初指向堆中的 Obj2 对象，当执行 O2=O1 后，O2 就指向了 O1，这时，从根集合出发，没有路径可以到达 Obj2,这时，Obj2 就成了垃圾对象，可以被回收。这里有一个问题，即然 Obj2 对象已经是垃圾对象，那么 JVM 何时会对它进行回收呢？要弄清楚这个问题，必须进一步详细了解垃圾回收的具体机制。

2.1.1 垃圾回收机制

首先，必须先了解一下堆内存的概念。堆内存是 JVM 管理的一种内存类型，JVM 管理的内存类型有两种：堆内存与栈内存。堆内存是用来存放对象的实例的，也就是说任何由 NEW 方法产生的对象实例和数组，都将被放入堆中。一个 JVM 实例唯一对应一个堆，堆是被所有线程所共享的。栈内存是用来存储程序代码中

声明为静态或非静态的方法。栈内存是属于线程私有的，JVM 会为每个正在运行的线程一个栈，而 JVM 对栈的操作是以桢为单位进行的，桢中保存的私有变量，操作数等

堆内存在 JVM 启动的时候被创建，堆内存中所存储的对象可以被 JVM 自动回收，不能通过其他外部手段回收，也就是说开发人员无法通过添加相关代码的手段回收位于堆内存中的对象。只能由 JVM 通过特定的垃圾回收机制来进行回收。常用的垃圾回收机制有以下几种：

(1)标记一清除收集器

这种收集器首先遍历对象图并标记可到达的对象，然后扫描堆栈以寻找未标记对象并释放它们的内存。这种收集器一般使用单线程工作并停止其他操作。

(2)标记一压缩收集器

有时也叫标记一清除一压缩收集器，与标记一清除收集器有相同的标记阶段。在第二阶段，则把标记对象复制到堆栈的新域中以便压缩堆栈。这种收集器也停止其他操作。

(3)复制收集器

这种收集器将堆栈分为两个域，常称为半空间。每次仅使用一半的空间，jvm 生成的新对象则放在另一半空间中。gc 运行时，它把可到达对象复制到另一半空间，从而压缩了堆栈。这种方法适用于短生存期的对象，持续复制长生存期的对象则导致效率降低。

(4)增量收集器

增量收集器把堆栈分为多个域，每次仅从一个域收集垃圾。这会造成较小的应用程序中断。

(5)分代收集器

这种收集器把堆栈分为两个或多个域，用以存放不同寿命的对象。jvm 生成的新对象一般放在其中的某个域中。过一段时间，继续存在的对象将获得使用期并转入更长寿命的域中。分代收集器对不同的域使用不同的算法以优化性能。

(6)并发收集器

并发收集器与应用程序同时运行。这些收集器在某点上（比如压缩时）一般都不得不停止其他操作以完成特定的任务，但是因为其他应用程序可进行其他的后台操作，所以中断其他处理的实际时间大大降低。

(7)并行收集器

并行收集器使用某种传统的算法并使用多线程并行的执行它们的工作。在多 cpu 机器上使用多线程技术可以显著的提高 java 应用程序的可扩展性。

2.1.2JVM 调优

下面介绍 sun hotspot1.4.1 使用分代收集器，它把堆分为三个主要区域[5]：新域，旧域以及永久域。Jvm 生成的所有新对象放在新域中。一旦对象经历了一定数量的垃圾收集循环后，便获得使用期并进入旧域。在永久域中 jvm 则存储 class 和 method 对象。就配置而言，永久域是一个独立域并且不认为是堆的一部分。

下面介绍如何控制这些域的大小。可使用-Xms 和-Xmx 控制整个堆的原始大小或最大值。

下面的命令是把初始大小设置为 128M：

java –Xms128m

–Xmx256m 为控制新域的大小，可使用-XX:NewRatio 设置新域在堆中所占的比例。

下面的命令把整个堆设置成 128m，新域比率设置成 3，即新域与旧域比例为 1：3，新域为堆的 1/4 或 32M：

java –Xms128m –Xmx128m

–XX:NewRatio =3 可使用-XX:NewSize 和-XX:MaxNewsize 设置新域的初始值和最大值。

下面的命令把新域的初始值和最大值设置成 64m:

java –Xms256m –Xmx256m –Xmn64m

永久域默认大小为 4m。运行程序时，jvm 会调整永久域的大小以满足需要。每次调整时，jvm 会对堆进行一次完全的垃圾收集。

默认状态下，HotSpot 在新域中使用复制收集器。该域一般分为三个部分。第一部分为 Eden，用于生成新的对象。另两部分称为救助空间，所有新创建的对象都在 Eden 区中分配空间，当 Eden 充满时，JVM 就要做可达性测试，主要任务是检测有哪象对象由根集合出发是不可到达的，这些对象就可以被 JVM 回收，并把所有可到达对象复制到当前的 from 救助空间，一旦当前的 from 救助空间充满，JVM 则会再次做可达性测试，并把可到达对象复制到当前的 to 救助空间。From 和 to 救助空间互换角色。维持活动的对象将在救助空间不断复制，直到它们获得使用期并转入旧域。使用-XX:SurvivorRatio 可控制新域子空间的大小。

联同 NewRation 一样，SurvivorRation 规定某救助域与 Eden 空间的比值。比如，以下命令把新域设置成 64m，Eden 占 32m，每个救助域各占 16m：

java -Xms256m -Xmx256m -Xmn64m -XX:SurvivorRation =2

默认状态下 HotSpot 对新域使用复制收集器，对旧域使用标记－清除－压缩

---

⑤林胜利，王坤茹，孟海利，《JAVA 优化编程》电子工业出版社，2002, 101-107

收集器。在新域中使用复制收集器有很多意义，因为应用程序生成的大部分对象是短寿命的。理想状态下，所有过渡对象在移出 Eden 空间时将被收集。如果能够这样的话，并且移出 Eden 空间的对象是长寿命的，那么理论上可以立即把它们移进旧域，避免在救助空间反复复制。但是，应用程序不能适合这种理想状态，因为它们有一小部分中长寿命的对象。最好是保持这些中长寿命的对象并放在新域中，因为复制小部分的对象总比压缩旧域廉价。为控制新域中对象的复制，可用-XX:TargetSurvivorRatio 控制救助空间的比例（该值是设置救助空间的使用比例。如救助空间为 1M，该值 50 表示可用 500K）。该值是一个百分比，默认值是 50。当较大的堆栈使用较低的 sruvivorratio 时，应增加该值到 80 至 90，以更好利用救助空间。用-XX:maxtenuring threshold 可控制上限。

为防止所有的复制全部发生以及希望对象从 eden 扩展到旧域，可以把 MaxTenuring Threshold 设置成 0。设置完成后，实际上就不再使用救助空间了，因此应把 SurvivorRatio 设成最大值以最大化 Eden 空间，设置如下：

java ... -XX:MaxTenuringThreshold=0 –XX:SurvivorRatio＝50000 ...

2. 2JAVA 中的内存泄漏

从上面对 JAVA 垃圾回收机制的分析中可以看出，JAVA 对内存的管理完全是由虚拟机自动管理的，不需要开发人员的人工干预，但是并不意味着垃圾回收机制是万能的，因为在堆中分配的对象只要还被引用，那么垃圾回收机制就不会对它进行回收。实际上，如果开发人员稍有不慎，会造成一些对象虽然不再需要，但是对它的引用并没有释放，这些仍然被引用但是实际上已经不再使用的对象就会造成内存的泄漏。下面，我们就可以描述什么是内存泄漏。在 Java 中，内存泄漏就是存在一些被分配的对象，这些对象有下面两个特点，首先，这些对象是可达的，即在有向图中，存在通路可以与其相连；其次，这些对象是无用的，即程序以后不会再使用这些对象。如果对象满足这两个条件，这些对象就可以判定为 Java 中的内存泄漏，这些对象不会被 GC 所回收，然而它却占用内存。
在 C++中，内存泄漏的范围更大一些。有些对象被分配了内存空间，然后却不可达，由于 C++中没有 GC，这些内存将永远收不回来。在 Java 中，这些不可达的对象都由 GC 负责回收，因此程序员不需要考虑这部分的内存泄露。

通过分析，我们得知，对于 C++，程序员需要自己管理边和顶点，而对于 Java 程序员只需要管理边就可以了（不需要管理顶点的释放）。通过这种方式，Java 提高了编程的效率。

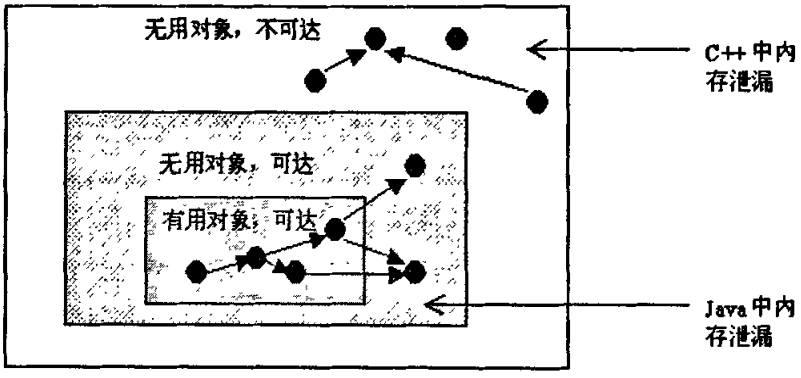

图 2.2 垃圾对象示意图

资料来源：http://favo.dbk2008.com/abstract/1000001570.php.html

因此，通过以上分析，我们知道在 Java 中也有内存泄漏，但范围比 C++要小一些。因为 Java 从语言上保证，任何对象都是可达的，所有的不可达对象都由 GC 管理

对于程序员来说，GC 基本是透明的，不可见的。虽然，我们只有几个函数可以访问 GC，例如运行 GC 的函数 System.gc()，但是根据 Java 语言规范定义，该函数不保证 JVM 的垃圾收集器一定会执行。因为，不同的 JVM 实现者可能使用不同的算法管理 GC。通常，GC 的线程的优先级别较低。JVM 调用 GC 的策略也有很多种，有的是内存使用到达一定程度时，GC 才开始工作，也有定时执行的，有的是平缓执行 GC，有的是中断式执行 GC。但通常来说，我们不需要关心这些。除非在一些特定的场合，GC 的执行影响应用程序的性能，例如对于基于 Web 的实时系统，如网络游戏等，用户不希望 GC 突然中断应用程序执行而进行垃圾回收，那么我们需要调整 GC 的参数，让 GC 能够通过平缓的方式释放内存，例如将垃圾回收分解为一系列的小步骤执行，Sun 提供的 HotSpot JVM 就支持这一特性。下面给出了一个简单的内存泄露的例子。在这个例子中，我们循环申请 Object 对象，并将所申请的对象放入一个 Vector 中，如果我们仅仅释放引用本身，那么 Vector 仍然引用该对象，所以这个对象对 GC 来说是不可回收的。因此，如果对象加入到 Vector 后，还必须从 Vector 中删除，最简单的方法就是将 Vector 对象设置为 null。

```
Vector v=new Vector(100);
for(int i=1;i<100; i++)
{
Object o=new Object();
v.add(o);
o=null;
```

```
}
```

此时，所有的 Object 对象都没有被释放，因为变量 v 引用这些对象。

2.2.1 如何预防内存泄漏

为了提高垃圾回收的效率，在实际应用中发现，以下几种方法可以减少或避免 JAVA 中的内存泄漏。

(1)慎用 System.gc()

在程序中显示的调用 System.gc()，这种方法并不一定保证执行 GC，因为不同的 JVM 实现者可能使用不同的算法管理 GC，有的是内存积累到一定程序的时候执行，有的定时执行，有的平缓执行，有的中断执行，有的可能因为当前系统内存能够满足需要而不会执行。另外，虽然 GC 的优先级较低，但其本身也是线程，也消耗系统资源，触发 GC 可能会造成间歇性停顿。

（2）减少临时对象的引用，在不用时最好显示的置为 NULL

临时对象使用后会成为垃圾，减少垃圾的产生，就减少了 GC 的负荷，对象不用时，最好显示地置为 NULL，防止内存泄漏，提高 GC 的效率。如在上面的程序中，要想让 GC 回收那些无用的对象，在使用完毕对象后，必需要将 Vector 对该对象的引用清除掉。如下：

```
Vector v=new Vector(100);
for(int i=1;i<100; i++)
{
Object o=new Object();
v.add(o);
……
o=null;
v.clear()//清除 Vector 对对象的引用
}
```

（3）尽量少用静态对象变量，慎用内部类

静态变量属于全局变量，不会被 GC 回收，会一直占用内存。非静态内部类中，隐含着外部类的对象实例的一个引用，这个引用无法消除。

（4）尽量少用 finalize 函数，不用过多的使用线程。

GC 在回收每个垃圾对象时，都会调用该对象的 finalize 函数，在 finalize 函数中的任何动作，都会增加 GC 的开销。线程越多，GC 挂起和恢复的时间就越长。

（5）按实际运行环境调整 JVM 的参数

GC 受系统内存环境因素的影响较大，所以可根据实际内存应用环境进行参数调整，测试。确定最优的参数值。

2.2.2 检测内存泄漏

在一个实际的 JAVA 应用中，特别是三层体系的 J2EE 应用当中，经常会出现 OutofMemory(内存不足)的错误，最终会导致系统崩溃，这种情况往往就是由于内存泄漏造成的，那么如何检测到内存泄露情况的发生呢，笔者总结出两个步骤。第一步，判断应用是否存在内存泄露，第二步分析和定位。

(1)如何判断应用是否存在内存泄漏。一般说来，一个正常的系统在其运行稳定后其内存的占用量是基本稳定的，不应该是无限制的增长的，同样，对任何一个类的对象的使用个数也有一个相对稳定的上限，不应该是持续增长的，基于这样的假设，我们可以持续的观察系统运行时使用的内存大小和各实例的个数，如果内存的大小持续增长，则说明系统存在着内存泄漏，如果某个类的实例个数持续增长，则说明这个类的实例可能存在泄漏。我们可以通过 JProfiler 观察内存的变动情况，JProfiler 是 Borland 公司开发的一款用于协助软件系统进行代码优化和诊断的工具，在实践中，可以分别在系统运行四个小时、八个小时、十二个小时和二十四个小时时间点记录当时的内存状态(即抓取当时的内存快照，是工具提供的功能，这个快照也是供下一步分析使用)，找出实例个数增长的前十位的类，记录下这十个类的名称和当前实例的个数。在记录完数据后，点击 Profiler 中右上角的 Mark 按钮，将该点的状态作为下一次记录数据时的比较点。

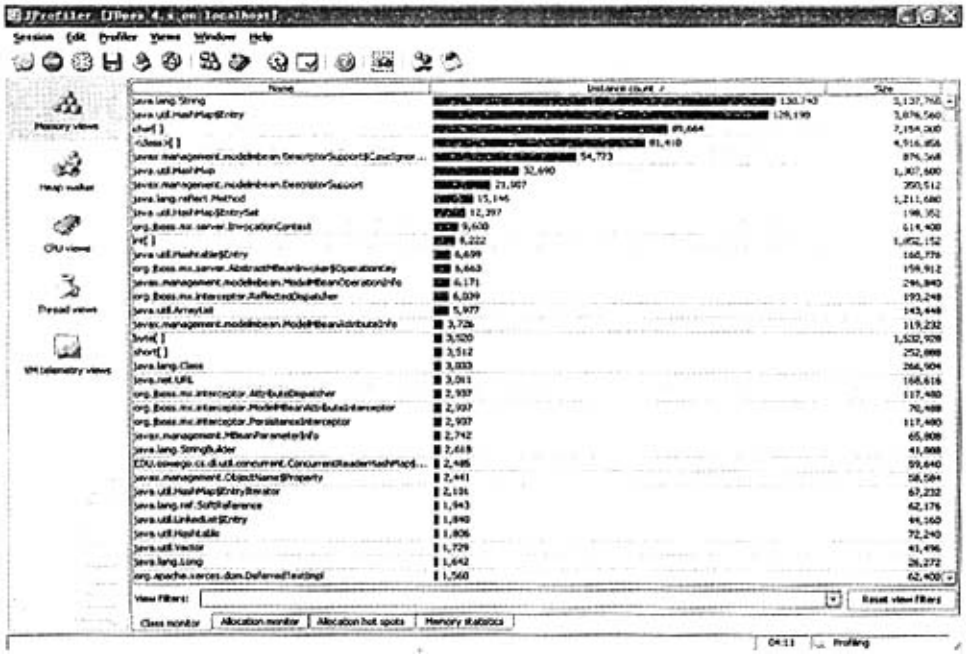

图 2.3 Profiler 堆视图

资料来源：本研究整理

系统运行二十四小时以后可以得到四个内存快照。对这四个内存快照进行

综合分析，如果每一次快照的内存使用都比上一次有增长，可以认定系统存在内存泄漏，找出在四个快照中实例个数都保持增长的类，这些类可以初步被认定为存在泄漏。

(2)分析与定位

通过上面的数据收集和初步分析，可以得出初步结论:系统是否存在内存泄漏和哪些对象存在泄漏(被泄漏)，如果结论是存在泄漏，就可以进入分析和定位阶段了。

前面已经谈到 Java 中的内存泄漏就是无意识的对象保持，简单地讲就是因为编码的错误导致了一条本来不应该存在的引用链的存在(从而导致了被引用的对象无法释放)，因此内存泄漏分析的任务就是找出这条多余的引用链，并找到其形成的原因。前面还讲到过根结点，包括已经加载的类的静态变量和处于活动线程的堆栈空间的变量。由于活动线程的堆栈空间是迅速变化的，处于堆栈空间内的牵引对象集合是迅速变化的，而作为类的静态变量的牵引对象的集合在系统运行期间是相对稳定的。

对每个被泄漏的实例对象，必然存在一条从某个根结点出发到达该对象的引用链。处于堆栈空间的引用对象在被从栈中弹出后就失去其引用的能力，变为无引用对象，因此，在长时间的运行后，被泄露的对象基本上都是被作为类的静态变量的引有对象引用。

Profiler 的内存视图除了堆视图以外，还包括实例分配视图(图 2.4)和实例引用图(图 2.5)。

Profiler 的实例引用图为找出从牵引对象到泄漏对象的引用链提供了非常直接的方法，其界面的第二个栏目中显示的就是从泄漏对象出发的逆向引用链。需要注意的是，当一个类的实例存在泄漏时，并非其所有的实例都是被泄漏的，往往只有一部分是被泄漏对象，其它则是正常使用的对象，要判断哪些是正常的引用链，哪些是不正常的引用链(引起泄漏的引用链)。通过抽取多个实例进行引用图的分析统计以后，可以找出一条或者多条从牵引对象出发的引用链，下面的任务就是找出这条引用链形成的原因。

实例分配图提供的功能是对每个类的实例的分配位置进行统计，查看实例分配的统计结果对于分析引用链的形成具有一定的作用，因为找到分配链与引用链的交点往往就可以找到了引用链形成的原因，下面将具体介绍。

图 2.4 实例分配图

资料来源：本研究整理

图 2.5 实例引用图

资料来源：本研究整理

设想一个实例对象 a 在方法 f 中被分配，最终被实例对象 b 所引用，下面来分析从 b 到 a 的引用链可能的形成原因。方法 f 在创建对象 a 后，对它的使用分为四种情况:1、将 a 作为返回值返回;2、将 a 作为参数调用其它方法;3、在方法内部将 a 的引用传递给其它对象;4、其它情况。其中情况 4 不会造成由 b 到 a 的引用链的生成，不用考虑。下面考虑其它三种情况:对于 1、2 两种情况，其造成的结果都是在另一个方法内部获得了对象 a 的引用，它的分析与方法 f 的分析完全一样(递归分析);考虑第 3 种情况:1、假设方法 f 直接将对象 a 的引用加入到对象 b，则对象 b 到 a 的引用链就找到了，分析结束;2、假设方法 f 将对象 a 的引用加入到对象 c，则接下来就需要跟踪对象 c 的使用，对象 c 的分析比对象 a 的分析步骤更多一些，但大体原理都是一样的，就是跟踪对象从创建后被使用的历程，最终找到其被牵引对象引用的原因。

现在将泄漏对象的引用链以及引用链形成的原因找到了，内存泄漏测试与分析的工作就到此结束，接下来的工作就是修改相应的设计或者实现中的错误了

2.3 本章小结

本章首先研究了 JAVA 的内存管理机制，从 JAVA 的内存分配与回收出发，探讨了 JAVA 内存泄漏出现的原因和危害，最后提出了如何避免 JAVA 内存泄露和检测内存泄露的方法。

第三章 EJB 性能优化

我们知道，J2EE 做为一种企业级开发技术，采用的是一种多层的体系结构，其基本的架构可分为 WEB 应用层、中间件层、数据库服务层，在这个基础上，又可以进一步的细分，形成多层的结构。J2EE 的优化技术涉及面很广，每一层都有值得研究的地方，本章只对 J2EE 的核心部分——中间件层的优化技术做一些探讨。

中间件层一般来讲，指的是 EJB（Enterprise java bean）以及 EJB 的容器。中间件层的优化对于 J2EE 应用的性能提升来说起着至关重要的作用。对中间层进行优化，主要应通过两种手段，一是池机制，一是负载均衡。下面两章分别对这两种技术进行研究。

3.1EJB 技术的优势

3.1.1EJB 技术的特性

EJB 通过提供下面的内建服务[⑥]，简化了开发人员的软件开发工作。

(1)它提供了 RMI/RMI-IIOP 协议（分布式计算），为实现分布式计算提供了协议支持。

(2)它提供了完善的安全机制。

(3)它具备出色的事务管理，使开发人员从烦琐，复杂的事务管理中脱身而出。

(4)它提供了性能卓越的资源池

(5)它是线程安全的。

(6)它可以保存客户端的状态。(stateful session EJB)

(7)它可以进行数据持久化操作。(Entity EJB)

(8)它可以进行消息传递、接受与处理操作(Message Driven Bean)

(9)它有自己的生命周期

3.1.2EJB 与传统 Bean 相比的性能优势

EJB 与传统的 Bean 相比，最大的特点是 EJB 采用了对象池的机制。如果要调用一个传统的 Bean,我们首先要将其实例化，然后再通过这个 Bean 的实例对象来引用这个 Bean 的属性和方法。如下面的代码所示：

```
Bean bean=new Bean();
Bean.getName();
String name=Bean.name;
```

⑥bill venners,《深入JAVA虚拟机》 机械工业出版社 2003,143-145

这种访问方法，在一般的中小型应用中是没有问题的，但是，试想在一个访问量巨大的应用中，要不断的实例化这个 Bean，而实例化的过程实际上是非常耗时和消耗资源的。那么频繁的实例化 Bean,不可避免的就会引起系统性能的下滑。

针对这个问题，EJB 技术提供了一个成熟的性能解决方案，对象池技术来解决这个问题。对象池技术就是在系统启动时就预先创建一些对象，在需要时，就从这些已创建的对象中借用一个，使用完毕后又放回去，这样就无须在每次使用时都要进行类的实例化操作，节约了系统开销，提高了系统性能。

再者，EJB 容器不但在单机中提供了对象池和缓存，而且可以跨服务器实现动态负载均衡，开发人员可以通过访问相应的 JNDI 资源来调用不同 EJB 容器中的 EJB 实例，来达到均衡系统负载的目的。

最后，作为一种标准的中间件，EJB 是可重用的，也就是说将已经开发好的 EJB 发布到任何适应的应用服务器上，这是普通的 Java Bean 难以做到的。

3.2EJB 的性能调整

3.2.1EJB 的类型

按照其自身特性的不同，EJB 可以分为三种[⑦]：会话 EJB(Session Bean) 实体 EJB(Entity Bean)和消息 EJB(Message Driven Bean)。

(1)会话 EJB

会话 EJB(Session Bean)的主要目的是可以令开发人员将商业逻辑层与数据层抽离，这样可以使开发人员只专注于相应的商业逻辑的开发，特别是相对复杂、多变的商业逻辑可以被放在会话 EJB 中。在会话 EJB 的家族中又可以根据其是否对应客户端的状态，还可以再细分为有状态会话 EJB(Sessionful Session Bean)与无状态会话 EJB(Stateless Session Bean)。它们的不同是一个使用者对应一个有状态会话 EJB 的实例。有状态会话 EJB 负责记录使用者的信息。而无状态会话 EJB 则不记录使用者的信息，也就是说实例池中的任意一个无状态 EJB 对使用者来说都是一样的，因此，无状态 EJB 在性能与维护的方便程度上都要优于有状态的 EJB。

(2)实体 EJB

实体 EJB(Entity Bean)是用来存储、提供数据信息的。实体 EJB 是用来描述关系型数据库中的一个数据表对象，其实例则代表关系型数据库中数据表的一行记录。实体 EJB 的主要目的是为商业逻辑层提供数据信息，也就是说实体 EJB 本身就是一个数据组件。在通常情况下，它不用来封装商业逻辑。至于实体 EJB

⑦吉庆洋，《JAVA内存泄露》

http://www.blogchinese.com/06081/242496/archives/2006/200691118517.shtml

是怎么存取实际上的数据库数据的，商业逻辑层也不用考虑过多。这样就有效地使数据层与商业逻辑层分离开来。

(3) 消息 EJB

消息 EJB(Message Driven Bean)与会话 EJB 或是实体 EJB 均不相同，它是 JMS 技术与 EJB 技术相结合的产物，它同时具备这两项技术的优点与特点。消息 EJB 是由消息触发、驱动的，会话 EJB 或者实体 EJB 都是由请求者发出请求来触发的，因此，请求者就是这两种 EJB 的驱动者，而且这种触发是同步完成的。但是消息 EJB 是由消息驱动、触发的，因此，在通常情况下这个过程是异步完成的。消息 EJB 并不是直接响应请求者请求的，而是通过消息队列来触发其执行相应处理的。

3.2.2 本地 EJB 与远程 EJB 的性能比较

EJB 的一个重要特性是支持分布式处理，也就是说客户端和 EJB 不是处于一个 JVM 当中，它与客户端的通信是通过 RMI/IIOP－RMI 协议进行通信的。其中还包括对象的序列化、信息编集与反信息编集的过程，当客户端在远程的机器上调用这个 EJB 的逻辑处理方法时，会给系统带来较大的开销。如果想减少这种开销，那么就必须保证将客户端与 EJB 发布在同一个 JAVA 虚拟机空间下。在 EJB2.0 中提出了本地 EJB 的概念，用于解决客户端和 EJB 在一个 JVM 中时，客户端对 EJB 的调用问题。客户端调用本地 EJB 时，不是通过 RMI/IIOP-RMI 进行通信的，并且也不会采用传递引用的方式进行远程调用，因此，采用本地 EJB 比采用远程 EJB 可以获得更好的系统性能和更快的响应速度。所以，我们用 EJB 技术开发软件时，首先要确定是选用本地 EJB 还是选用远程 EJB，本地 EJB 有更好的性能，远程 EJB 可以支持分布式计算。在一般情况下，建议选用本地 EJB。

需要注意的是，本地 EJB 只能应用在 EJB2.0 的环境下。在 EJB1.1 的环境下是不支持本地 EJB 的，因此即使将远程 EJB 与客户端部署在同一个 JVM 下，但它们之间的通信仍然会采用 RMI/IIOP-RMI 协议，并且仍然通过传递引用的方式来完成对象的引用与方法的调用。这不能提高系统的性能，那么对于引用对象，怎么从传递引用的方式转化为传递对象值呢？一些应用服务器提供商提供了解决办法，在 JBOSS 服务器上，可以通过其部署文件 JBOSS.xml 中的下面的属性达到上述目的：

```
<container-invoker-conf>
    <optimized>true</optimized>
</container-invoker-conf>
```

3.2.3 优化无状态会话 EJB 的性能

上文已经提及，客户端是无法控制无状态会话 EJB 的生命周期的，它的生命周期完全由 EJB 容器控制和管理的。EJB 容器在应用服务器启动的时候，在

EJB 实例池中初始化创建的 EJB 的最小实例数和最大实例数。也就是说可以通过控制实例池中 EJB 的实例数来控制 EJB 的生命周期。在 JBOSS 应用服务器中，可以配置 EJB 发布描述文件 JBOSS.XML 文件，设置下面的属性来控制 EJB 实例池中的实例数量。

```
<instance-pool>
<max-beans-in-free-pool>100</max-beans-in-free-pool>
<initial-beans-in-free-pool>50</initial-beans-in-free-pool>
</instance-pool>
```

在应用服务器启动的时候，EJB 容器首先通过调用 Class.newInstance()方法创建 50 个 EJB 实例，并且将它们放在 EJB 实例池中，接着会回调这些 EJB 实例的方法，如下所示。

```
SetSessionContext(ctx);
EjbCreate();
```

初始化时创建的 50 个 EJB 实例，可以并行处理 50 个用户的访问。当无法满足新的用户请求的时候，例如，并发的用户达到 51 个。此时 EJB 容器会自动创建新的 EJB 实例来响应新到来的请求。直到 EJB 实例池中的 EJB 实例数量达到 100 时，EJB 容器将不会再创建新的 EJB 实例。

当并发的用户数据继续增长的时候，这些后来的用户只能处于一种等待的状态，只有当 EJB 实例池中有闲置的 EJB 实例时，才会响应这个用户，例如，当有第 101 个用户试图访问 EJB 实例时，因为此时已经有 100 个用户正在访问实例池中的实例，并且该 EJB 实例池中的数量已经达到上限。因此，此时系统不能及时地响应第 101 个用户的请求，只有等其中有一个用户的请求被处理完毕，有相应的 EJB 实例被释放到 EJB 实例池之后，系统才能响应到第 101 个用户的请求。当并发访问的用户数下降时，也就是说低于 100 个并发用户时，此时 EJB 容器会移除一些 EJB 实例，并且最低维持在 50 个 EJB 实例的数量上。在 EJB 容器移除每一个 EJB 实例的时候，容器都会主动地调用 EJB 实例的 ejbRemove() 方法。关于移除 EJB 实例的算法会因应用服务器的不同而不尽相同，通常来讲会采用最近最少使用(LRU)算法来处理。

从上面的分析可以看出，通过指定合适的 EJB 实例池中实例数量的值来提高应用性能的关键，是设置一个最优的最大值。如果设置的这个最大值较小的话，当新的客户端请求到来时，由于实例池中的 EJB 实例数据已经达到上限，此时客户端必须等待。因此，客户端的请求以及得到服务器响应的时候无疑将会变得很长。但是，这并不意味着将 EJB 实例池中实例数量的最大值设得越大越好，如果将这个值设得很大，但是在实际应用中，根本不可能用到这么多的实例，同样会影响到应用的性能。假如一个应用最多只有 100 个用户，而你却把最大值设

为 1000，那么将极大地消耗服务器端的系统资源，降低系统的性能。在通常情况下，可以采用这一法则来得到一个相对科学的数值。并发访问系统的最大峰值通常为实际用户数据的 1/50。也就是说一个实际拥有 1000 个用户系统，可能出现的实际并发访问系统的峰值会在 200 左右。但这只是一个理论值，并不是适合于任何系统。寻找最优的数值，还需要根据应用所面向的具体情况，经过科学的调研后才能得到。

当创建无状态会话 EJB 实例的时候，EJB 容器将调用 setSessionContext(SessionContext sc)方法与 ejbCreate()方法来做相应的初始化工作。在这个 EJB 实例被创建之后，它会一直等待来自客户端的请求，直到它被 EJB 容器移除。因此，可以借助于这两个方法来缓存一些会被其它用户用到的数据资源。例如，可以缓存一些远程引用的对象，数据源对象等。这些资源可能会被所有的系统用户，或者多个系统用户用到，而不需要每次都去重复地做相应的初始化工作，尤其是那些需要耗费较长时间方可取到的系统资源，将其缓存起来的必要性就变得更为明显。

3.2.4 优化有状态会话 EJB 的性能

上文已经提到，有状态会话 EJB 与无状态会话 EJB 的主要区别在于是否具有对客户端状态信息保存的能力[⑧]。应用服务器是采用 EJB 实例池的技术来缓存无状态会话 EJB 实例的。而对于有状态会话 EJB 而言，这个实例池已经被实例缓冲的技术概念所取代，实际上二者大同小异，其根本区别在于是否可以维护 EJB 实例的状态信息。

与控制无状态会话 EJB 的生命周期类似，同样也可以通过设置实例缓冲区的大小来控制有状态会话 EJB 的生命周期。以 JBOSS 应用服务器为例，需要修改 Jboss.xml 中的属性值来指定 EJB 实例缓冲区的大小，如下所示。

```
<instance-cache>
<container-cache-conf>
  <cache-policy>
        <cache-policy-conf>
            <min-capacity>50</min-capacity>
            <max-capacity>100</max-capacity>
        </cache-policy-conf>
    </cache-policy>
</container-cache-conf>
</instance-cache>
```

⑧约翰逊，《J2EE设计开发编程指南》，清华大学出版社, 2001, 156-160

与有状态会话 EJB 不同的时，这些实例并不能在应用服务器启动的时候就被创建，即使设置了 EJB 实例缓冲的大小。这些实例是由客户端调用方法 create()创建的。但是 JBOSS 服务器为了优化其性能，而是先创建了一些 EJB 实例，并且将它们放入 EJB 实例缓冲区中。此时，这些实例没有状态，或者说都具有相同的默认状态。EJB 容器却是可以管理 EJB 的状态，EJB 容器可以把一个 EJB 实例由活动状态置为休眠状态，同样也可以将它由休眠状态置为活动状态。

优化有状态会话 EJB 的性能，主要通过四个方面来考虑：设置合适的 EJB 实例缓冲区的大小；控制序列化对象的数量；合理设置 EJB 实例的寿命；显示移除 EJB 实例。下面分别介绍。

(1)设置合适的 EJB 实例缓冲区的大小

设置合适的 EJB 实例缓冲区的大小，对提高应用的性能是非常重要的。如果将实例缓冲区大小的值设置得较小，那么当客户端在实例缓冲区满后发送过来的请求，将会被压入等待的队列，而不能及时响应客户端的请求，造成阻塞现象。但是如果这个值设得过大，又会带来浪费系统内存，降低系统性能的问题。因此，应该根据实际需要，综合权衡各种因素为其设置一个最为合适的值。

(2)控制序列化对象的数量

简单而言，序列化对象[⑨]就是实现了可序列化接口的 JAVA 类的实例。当有状态会话 EJB 被由活动状态置为休眠状态的时候，它要把这个 EJB 实例存储到一个相应的位置，同时还要把这个实例所包含的序列化数据也存储在相应的位置。当有状态会话 EJB 被由休眠状态置为活动状态时，同时还需要做反序列化操作，把存储在相应位置上的序列化数据还原给这个会话 EJB 实例。如果一个有状态会话 EJB 实例中的序列化数据较多，在反复执行上述操作的时候，将会消耗较多的系统资源，同时会降低系统性能。因此，在实际的开发应用中，最好不要在有状态会话 EJB 中放置过多的序列化对象，以免影响系统的整体性能。

(3)合理设置有状态侍话 EJB 实例的寿命

在一些应用服务器的配置文件中，可以设置有状态会话 EJB 实例的寿命。这个值设得较小时，则会引起频繁调用 create() 方法与 remove()方法的问题。而这两个操作又是非常耗时的，因此，这会引起系统性能的下降。理论上讲，这个值设得越大越好。

(4)显式移除 EJB 实例

如果有状态的会话 EJB 的实例长时间不使用，则会被 EJB 容器置为休眠状态。并且一直维护这个实例，直到这个实例被移除。因而必将浪费大量的没有任何意义的内存资源。但是可以通过客户端显示地调用 remove()方法来主动地移除

⑨ Martin bond, Dan Haywood, Debbie Law and Peter Roxburgh "Teach yourself in 21 days" Sams , 2002.

这个已经不再被启用的 EJB 实例。这样可以在一定程序上释放不必要的内存资源，提升系统的性能。

3.3 本章小结

本章首先比较了 EJB 技术与普通 javabean 的性能优势。并且进一步指出本地 EJB 与远程 EJB 的不同，最后，详细阐述了如何在 JBOSS 平台下，利用 EJB 的池机制来调整应用的性能。

第四章 J2EE 集群技术分析

前面几个章节我们主要研究的是如何通过优化代码和调整服务器的配置来提高效率，但是在一个大型的J2EE应用中，仅仅依靠上述的工作，来支撑大量的用户访问显然是不高的。同时对于大型的J2EE应用来说，除了效率外，高可用性与高可扩展性也是必不可少，那么如何满足应用对高可用性和高可扩展性的要求呢？目前通用的做法是使用J2EE集群技术，本章将对JBOSS平台实现的J2EE集群技术进行研究，并将从实践中验证集群技术的有效性。

4.1J2EE 集群

4.1.1 集群的概念

集群[9]是对客户提供统一服务的松散连接的一组服务器。使用部署在集群中的服务的客户通常感觉不到他们的请求是通过集群实现的，并且客户不能指定由哪一台服务器来处理它的请求。在集群中的服务器可能使用一台或多台计算机，每台计算机有一个或多个处理器。集群利用高速通用网络将一组高性能工作站或高档 PC 机，按某种结构连接起来，并在并行程序设计以及人机交互集成开发环境支持下，统一调度，协调处理、实现高效并行处理的系统。从结构和节点间的通信方式来看，它属于分布存储系统，主要利用消息传递方式实现各主机间的通信，由建立在一般操作系统之上的并行编程环境完成系统的资源管理及相互协作，同时也屏蔽工作站及网络的异构性，对程序员和用户来说，集群系统是一个整体的并行系统。

通常来讲，J2EE 集群主要实现了两种功能：“负载均衡”和“失败转移“。负载均衡就是指，在同一时刻，有很多客户端对目标对象进行请求，负载均衡器就位于调用器和被调用者之间，它将请求分发到具有同样功能的冗余对象中去。

图 4.1 负载均衡

资料来源：http://www.javaresearch.org/download/41616.htm

[10]Sun Micro systems, Java2 platform Enterprise Edition1, 3 Specfication, Public draft, 2000:22-57

失败转移[11]和负载均衡的工作方式有所不同。通常情况下，客户端对象的调用都是成功的，当服务器端的被调用对象失败之后，失败转移系统就能够检测到这个失败，并且把后继的请求转发到冗余的、可用的对象。这样就达到了容错的目的。

图4.2 失败转移

资料来源：http://www.javaresearch.org/download/41616.htm

4.1.2 集群的范围

并不是所有的对象都可以被集群。为了能够进行负载均衡和失败转移，在调用者和被调用者之间必须要有一个拦截器，这个拦截器能够将调用请求转移到其它的对象。也就是说调用在不同的 JVM 中的对象时，才能进行负载均衡和失败转移。分布式组件是 J2EE 的重要特征之一，组件分布在不同的容器中，因此，理论上说，一个 J2EE 的应用中，只要是两个位于不同容器中的组件进行交互时，都可以进行负载均衡和失败转移。

11Altendorf, EHohman, Zabicki, Using J2EE on a large Web-based project, Software, IEEE, Volume, 2002, 3-5:81-89

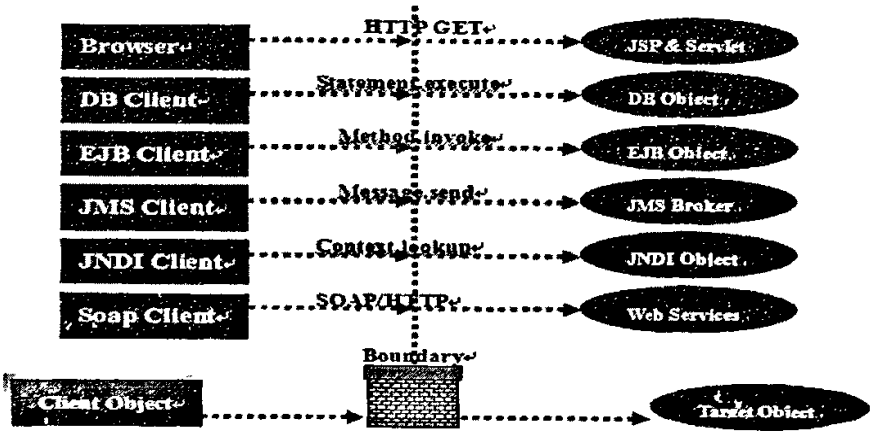

图4.3 集群的范围

资料来源：http://www.javaresearch.org/download/41616.htm

在一个分布式环境中，如上图所示，调用者和被调用者在不同的运行容器中，并且有明显的分界线将他们分隔开来，例如不同的 JVM，不同的主机，或者不同的进程。当目标对象被客户端调用的时候，目标对象运行在自己的容器中。客户端和对象服务器之间通过常见的网络协议进行通信。这种特点就使得有机会对于调用的路由进行干预，从而实现负载均衡和失败转移。如上图中所示，浏览器通过 HTTP 协议请求远端的一个 JSP，这个 JSP 在 WEB 服务器中运行。对于浏览器而言，它只关系结果，不关心如何执行的。在这种情况下，就可以在调用者和被调用者之间插入一些东西来实现负载均衡和失败转移。在 J2EE 中，分布式技术包括：JSP/Servlet，JDBC，EJB，JNDI，JMS，Web Services 等。这些分布式调用中，可以进行负载均衡和失败转移。

我们可以这样描述一个 J2EE 典型应用的调用过程，一个用户通过浏览器向 WEB 应用服务器发出请求，WEB 服务器根据接到的请求，调用相应的 EJB 容器的 EJB 组件，并将处理后的结果返回给客户端。从以上描述的过程中，我们可以看出，在一个 J2EE 应用的调用中有两个明显的分界线，一是用户对 WEB 服务器进行调用时，我们可以在用户和 WEB 服务器之间加入负载均衡的逻辑，将用户的请求按照一定的规则分配到集群中的一个 WEB 服务器进行处理。第二条分界线是 WEB 服务器在调用 EJB 时，WEB 服务器根据规则从 EJB 集群中选择一个，进行业务逻辑的处理。

从以上的分析可以看出，J2EE 的集群主要应分为两个部分，WEB 层集群和 EJB 层集群，下面会对它们分别描述。

4.2WEB层集群技术

WEB层的集群技术包括WEB层的负载均衡和HTTP会话的转移失败。

4.2.1WEB负载均衡技术

一般来说，是在浏览器和WEB服务器之间设置一个负载均衡器[12]来对客户的请求进行分发。负载均衡器可以是一个硬件，也可以是软件，下面是几种常用的WEB层负载均衡的实现方式。

(1)DNS循环

DNS负载均衡是一种简单而有效的方法，该方法使用简单的域名查找IP地址来实现一种简单的负载均衡。任意给出一个地址，DNS服务器都有一个IP地址池与之对应。每次请求将域名转换成IP地址时，循环返回IP地址池中的下一个地址。故被称作DNS round-robin。当一个Client访问时，给请求JNDI的InitialContext客户端传递一个DNS名，作为命名服务器的URL，每个DNS名字被转换成一个不同的地址，使用这个技术，每个客户端InitailContext请求就被直接发送到不同的服务器上。此种负载均衡的一大缺点是:一旦某个服务器出现故障，即使及时修改了DNS设置，还是要等待足够的时间(因为DNS需要一定的刷新时间)才能发挥作用，在此期间，有些客户端用户访问仍旧将发送故障服务器上。

(2)软件Proxy

软件Proxy维护连接到一系列服务器上的打开连接。当一个Client访问服务器时，先要经过这个软件代理，这个代理能通过一些负载均衡的算法(如采用类似DNS Round-r obin、随机方法、访问权衡算法)把一个用户的访问重新定向到一个服务器。这个软件代理方法能够及时发现服务器死机或没有响应，有效地避免了DNS round-robin方法中出现的故障访问

(3)硬件均衡器:

这种硬件均衡器一般采用地址转换技术，将一个外部IP地址映射为多个内部IP地址，对每次TCP连接请求动态使用其中一个内部地址，达到负载均衡的目的。一般可采用第四层(或4层以上)的交换机来实现，这种交换机是按照IP地址和TCP端口进行虚拟连接的交换，直接将数据包发送到目的计算机的相应端口。通过交换机就能将来自外部的初始连接请求，分别与内部的多个地址相联系，从而建立虚拟连接实现负载均衡。这种第四层交换基于硬件芯片，因此网络传输速度和交换速度远远超过普通软件代理方式。如采用Cisco CSS 11150(一种L4 Switch)可以实现硬件均衡。

12肖辉，《应用服务器中间件EJB容器群集机制研究》，西安交大硕士论文，2003:8-9

4.2.2WEB层失败转移

WEB层的失败转移功能是建立在会话复制的基础上的。在J2EE的Web程序中的Jsp/Servlet中经常用到Session对象，它的主要功能是记录访问端客户会话中有关信息。在一般的Web系统中，在Client访问阶段，假设系统中Server端死机后，这次Client访问中Session中保留的相关信息将全部丢失。用户下次访问需要重新开始建立新的Session. HttpSession Cluster的目的就是即使服务器端系统死机重新启动，上次未完成访问中的Session在系统中仍旧保存，Client可以接着完成上次未完成的访问。一般实现HttpSession Cluster有下面两种方法:

(1)采用内存复制:

在这个方法中,当某台J2EE Server中的某个HttpSession或某个Object被修改后，或是新增一个Object后，这台Server可以采用某种通讯方式(如IP Broadcast)将此Object传送到其它备份的J2EE Server上，让其它J2EE Server上相同HttpSession同步变化，例如，BEA WebLogic Server实现了session信息的内存复制。

基于内存的session持久性实现是通过把内存里的session信息存储到备份服务器。这种方法存在两种变化.第一个方法把HttpSession信息写到一个专门的状态服务器.群集里的所有机器把它们的HttpSession对象写到这个服务器.在第二种方法里,每个群集节点选择一个任意备份节点来存储内存里的session信息。用户每次增加一个对象HttpSession，那个对象独自被序列化，然后从内存里添加到一个备份服务器.

如果群集里的服务器数目小的话,这个专门的状态服务器证明了比内存复制到一个任意的备份服务器更好，因为它为事务处理和动态页面生成，释放了CPU周期。另一方面,当群集里的服务器实例数目较大时，这个专门的状态服务器会成为瓶颈，当增加更多的服务器时，内存复制到一个任意的备份服务器(相对于一个专门的状态服务器来说)将线形伸缩。此外,当在群集里增加更多的机器时，也需要增加更多的RAM和CPU来持续的调整这个状态服务器。而对于内存复制到一个任意的备份服务器来说，只需添加更多的机器，sessions可以均匀的把自己分布到群集里的所有机器上。总言之，基于内存的持久性提供了柔性的web应用设计，伸缩性好，并实现了高可用性.

(2)数据库/文件系统

将会话状态保存到数据库或文件系统中。SilverStream plicationServer和Sybase Enterprise Application Server在实现会话复制上利用了一个所有应用服务器都要读写的中央集中数据库或者文件系统。采用数据库/文件系统方法，session持久性的主要缺点集中在:当存储大的或者众多的HttpSession对象时受限的伸缩性.一个用户每次增加一个对象到HttpSession，在session里的所有对

象都被序列化写到一个数据库或者共享文件系统。大多数利用了数据库session持久性的应用服务器提倡最小限度的使用HttpSession存储对象，但是这限制了web应用的架构和设计，尤其是在使用HttpSession来存储缓存的用户数据。

有了会话状态的备份，就可以轻易的实现会话的失败转移。Http Session失效恢复逻辑主要包含在客户端cookie或url或客户端proxy中。状态和会话数据被复制到群集内其他服务器内存或是被存储在永久性媒介(如数据库)中，使用户数据对每个服务器总是保持可用。当客户端在原始的服务器上建立的session失败时，新的请求将被重新路由到其它服务器，允许客户端从群集里的其他服务器无缝的取得session信息。比如，Weblogic利用客户端的cookie来保存session副本所在的服务器列表，当失效时，从该列表中取出一可用服务器进行连接，如果cookie不可用，则通过url来传递Session副本所在的服务器列表。

4.3EJB 集群技术

EJB是J2EE技术中非常重要的一部分，同时，EJB的集群也是J2EE集群实现中面对的最大挑战。EJB是为分布计算而生的，他们运行在独立的服务器上，WEB服务器组件或者富客户端应用通过标准协议（RMI/IIOP）访问这些EJB，你可以像调用本地对象方法一样调用远程EJB对象的方法，实际上，RMI－IIOP[13]掩盖了你所调用的方法是在本地还是在远程，这个就叫做“本地/远程透明性“。

4.3.1EJB 的负载均衡

当客户端需要使用一个EJB的时候，不能直接调用这个EJB，客户端只能调用一个叫做“stub”的本地对象，这个本地的“stub”和远程的EJB有相同的接口，起到代理的作用，“stub“的作用就是接受客户端的调用，并且把这个调用通过网络委派到远程的EJB对象。Stub和客户端应用运行在同一个JVM实例中，它知道如何通过RMI/IIOP在网络上找到真正的对象。客户端调用EJB的基本步骤如下：

第一步，从一个 JNDI 服务器上找到 EJBHome stub.

第二步，使用 EJBHome stub 和 EJBHome skeleton 进行通信，由 skeleton 调用 EJBHome 的实现类来查找或者创建一个 EJB 对象，返回客户端一个 EJBObject stub。

第三步，通过 EJBObject stub 和 EJBObject Skeleton 进行交互，由 EJBObject Skeleton 调用 EJBObject 的实现类，并由 EJBObject 的实现类进行实际的 EJB 的方法调用，并将方法的执行结果传回客户端。

13陈耿眠，晏蒲柳，郭成城等，《基于Web内容的集群服务器请求分配系统的研究与实现》，计算机工程，2003(01):87-92

可以看出在EJB的调用过程中，理论上可以有两种方法实现EJB容器的群集，一种方法是通过复制JNDI树，并且在JNDI查找的时候完成系统的负载均衡，因为无法得到结点负载信息，所以只能使用简单的静态负载均衡方法，负载均衡效率不高。但这种方法的实现比较简单。另一种方法是首先在部署时由自动部署工具完成分布式复制部署，然后通过一个负载信息收集器来完成本地负载信息收集和管理，并且由状态管理器同步管理群集系统中的EJB的状态，而在EJBHome对象和EJBObject对象的Stub或Proxy中，包含必要的群集信息。这样当客户端进行方法调用的时候才会生成EJBObject对象的动态Proxy.再将包含负载均衡策略信息的动态Proxy由客户端动态下载到本地完成调用，从而可以通过这个方法来实现EJB容器群集中的负载均衡和容错。这种方法能够实现动态的负载均衡策略，并且具备对客户端的透明性，高可靠性等优点。

EJB负载均衡可分为两个级别:对象级和方法级。对象级是指通过全局JNDI树来查找EJBHome对象时，通过负载平衡器把请求分配到合适的结点，返回EJBObject对象，以后对于EJBObject的业务方法的调用，不再进行分配。对象级的负载均衡在一定程度上平衡系统的负载，负载平衡的消耗小，而且实现也比较容易，但是缺点也是很明显的，不能动态平衡系统负载。而方法级的负载平衡在每次EJBObject的方法调用时，都通过负载平衡器，进行请求分配，以实现系统负载的均衡，但是方法级的负载平衡更加复杂，需要同步bean的会话状态，在群集范围内传播事务上下文等等问题，所以负载平衡的消耗大，实现复杂。

4.3.2EJB 的失败转移

失效恢复要解决的问题是:home或remote的存根(stub)是如何检测到失败情况出现呢?如果一异常产生，存根将如何恢复失效情况?

存根可接收的有三种类型的异常:应用程序异常(由bean developer定义)，系统异常(典型地以EJBException形式出现)，network/communication异常。前两者在EJB specification中清楚地定义，并传到调用的客户端进行处理。第三者SocketException, CommunicationException和其它，在不跟服务器通信的情形下，属于不可恢复的的失败情况。应用服务器存根能截取(intercept)所有系统异常和通信异常，并能执行失效恢复，如果安全的话，将请求转到其它服务器。

EJB失效通常出现以下三种情况:

(1) 失效出现在服务器端调用开始前。如果存根能断定失败出现在方法请求已发送出去后而在服务器调用它之前，这个存根可以将这个失败看作是负载均衡情况，并将方法调用转到其它可用的服务器即可解决问题

(2) 失效出现在服务器调用结束后，但在响应信息传送回客户端之前。这种情况不需要存根进行任何操作。

(3) 失效出现在服务器端调用过程中。方法调用在服务器上被激发，有可

能已改变了服务器端的部分资源，这些修改也有可能对接下来的系统行为产生影响。如果存根对这一方法进行下一请求，且方法的调用会影响到服务器端的资源或数据的，这个存根在做失效恢复操作过程中有可能不注意的执行相同的修改操作两次，因而导致系统存在不确定因素。如果这方法的调用不执行任何修改等更新操作，它就可以安全地被重新调用多次，而不至于对系统产生不良影响。

困难的是，虽然a和b两种情况产生的结果乐观，但要决定系统在哪种状态却不容易。因此，在大多实现中，存根作如下假设:一旦出现了失效情况，都假定服务器处于c情况，作最坏的假想，即使它实际上是在a或b情况。为防止c情况下修改同一资源多次，导致某一service被请求了多次的错误结果，一般通过添加唯一标志加以控制。例如:客户端C请求Clustered购物车服务中的additem()方法，replica-aware stub接收到请求，根据算法调用Server1上的service，Server1响应请求并返回结果，但在结果成功到达客户端之前，Server1出现错误。此时stub接收到错误信息，因此重新调用Server2上的这一方法，但实际上Server1已经将item加入购物车，这样就造成重复。为了解决这种问题，可以为服务添加一个唯一标识，如上述的additem()方法中可添加一个参数——序列号。每一个item有一个唯一的sequence，相同sequence的item不能被重复添加。

大多应用服务器都提供了幂等[14]方法，即在一连串调用后，只要输入参数一样，返回结果是一样的。不幂等的方法调用会改变系统的状态，也就是在下次执行时会产生不同的结果。幂等的方法类型包括:

(1)所有reader/getter方法

(2)执行非修改性的查询

(3)访问一数据管理器，但不改变数据管理器的状态

前面提到过，一般在EJB对象的存根里包含了均衡逻辑和失效逻辑，至少它也包含该EJB对象的副本所在的服务器的列表，当失效出现时，按一定的算法，从该列表中取出下一可用的服务器，并由所选出的服务器接管失败。对无状态会话bean，只要对它的调用是幂等的，通过远程调用ejbCreate方法，即可恢复;对有状态会话bean，当主拷贝失败，访问备份;对实体bean，失效恢复从本地转到了远地接口调用。

4.4 几种不同的集群方式的比较

J2EE技术，特别是EJB是为分布式应用而生，松耦和的业务方法，可重用的远程组件使得多层应用渐成流行之势，但这并不意味着要把所有的东西都做成分布式的。因为从J2EE的技术基础来看，的EJB的调用是建立在RMI/IIOP通信协议

14骆宗阳，董玮文，杨宇航等，《一种具有高可用性的通用负载均衡技术》.计算机工程，2003(02):23-27

基础上的，要通过对象序列化的技术实现对象在客户端在服务端的传递，不幸的是，连J2EE规范的制订者SUN公司自己也承认RMI/IIOP通信协议以及对象序列化技术是一种效率并不高的技术，EJB的调用速度很容易成为分布式应用系统的瓶颈。让我们来看一下一个典型的J2EE多层应用的场景。

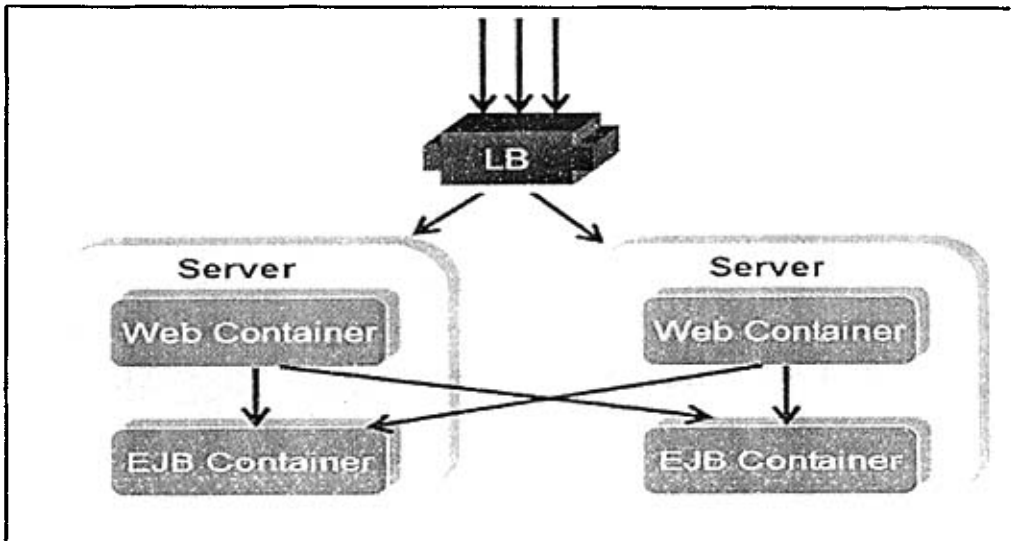

图4.4典型集群方式

资料来源：http://www.javaresearch.org/download/41616.htm

当请求到来的时候，负载均衡器将请求转发到不同服务器实例上的WEB容器。如果这个请求包含了对EJB的调用，那么这个对EJB的调用将会重新分发到不同的EJB容器。这样，请求就被负载均衡和失败转移了两次。

实际上，这样复杂的负载均衡和失败转移并不是最好的。存在着以下几个问题

(1)第二次负载均衡是没有必要的，因为它不能均匀分配任务。每个服务器都有自己的WEB服务器和EJB容器，让EJB容器来处理来自其它服务器WEB容器的请求相比于处理自己WEB服务器中的请求来说，没有显示出任何优点来。

(2)第二次的失败转移是没有必要的，因为它不能提高可用性。大部分服务器产品的WEB服务器和EJB容器都在同一个JVM实例中运行，如果EJB容器失败了，大部分情况下，WEB容器也失败了。

(3)降低了性能，假如在应用中的一个方法调用多个EJB，在这些EJB上进行了负载均衡，那么每次请求都被分散到不同的服务器实例，这样就产生了不必要的，跨服务器交互。并且，如果这个方法中有事务，那么这个事务的边界就需要包含多个服务器实例，这是非常影响性能的。

在实际的运行环境中，很多厂商（WEBLOGIC，JBOSS）对于EJB的负载均衡都做了优化，优先选择同一个服务器上的EJB。这样，负载均衡仅在第一层上进行（WEB容器）进行，接下来的请求就在同一个服务器上处理了。这叫做“搭配结

构”[15]。搭配结构属于分布式处理的一种特殊方式。

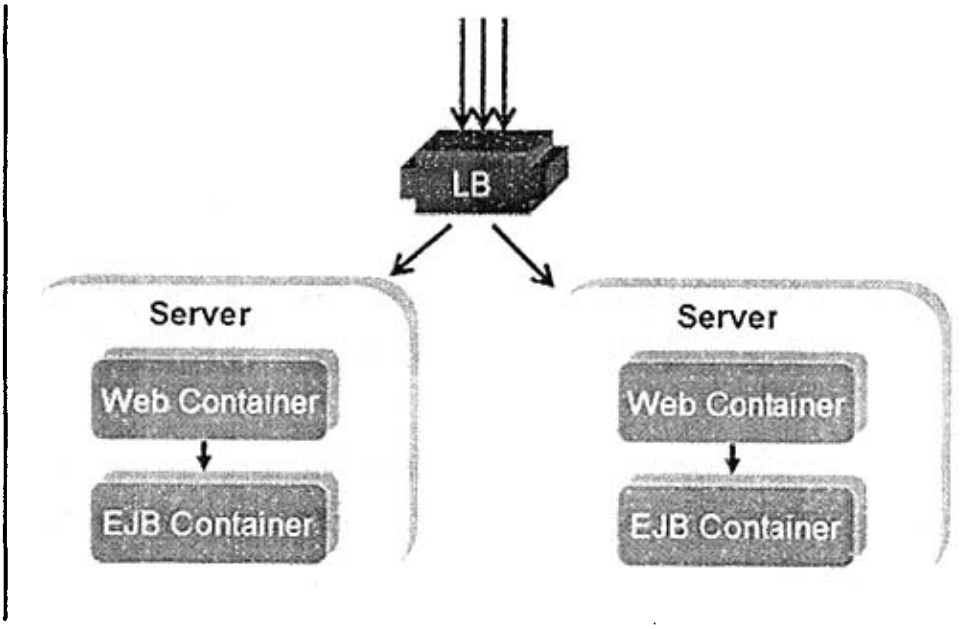

图4.5搭配式集群方式

资料来源：http://www.javaresearch.org/download/41616.htm

在搭配式结构下，虽然WEB服务器和EJB容器在部署在同一个主机上，而且同一个请求都是在同一主机上进行处理的，但是WEB容器和EJB容器仍然是处在不同的两个JVM之间，两者之间的通信仍然要通过RMI/IIOP通信协议进行，因此，对EJB的调用仍然是一个非常耗费资源的工作。

SUN公司自己也意识到了EJB在效率上存在的问题，自EJB2.0起，推出了本地接口EJB[16]，本地接口EJB是位于和WEB组件同一个JVM当中，因此，WEB组件对本地接口EJB的调用是直接调用，不再依赖RMI/IIOP协议。因此，EJB的效率得以大大提高。当然凡事有利必有弊，本地接口虽然提高了效率，但是使EJB与WEB组件成为紧耦合的，对EJB的调用就不能加入负载均衡和失败转移的功能了，只能对WEB+EJB做为一个整体来进行负载均衡和失败转移了。但是对大多数应用来说，在EJB层做负载均衡和失败转移所付出的性能代价往往是不能承受的，因此，对

15 P.K ruegerand, M .Livny.A comparison of Preemptivge and Non-Preemptive Load Distributing, Technical Report, Department of Computer Science, University of Wisconsin, Madison, WI, 1998:58-96

16 M.H .Mickle and J.M .Paul, Load Balancing Using Heterogeneous Processors for Continuum Problem on a Mesh, Journal of parallel and Distributed Computing, 1996:66-73

于一般的应用来说，WEB+本地接口EJB往往是J2EE应用的最佳选择。

4.5 本章小结

本章详细描述了 J2EE 集群的概念，说明了在 J2EE 应用中集群技术适用的范围。然后又分别对 WEB 层和 EJB 层的集群技术做了深入的探讨。最后，比较了几种集群方式的区别，得出对于一般的应用来说，WEB+本地接口 EJB 往往是 J2EE 应用的最佳选择的结论。

第五章 动态负载均衡的实现

在上一章中进行了几种集群方式的比较，可以看出 WEB+本地接口 EJB 的集群方式有较好的性能，也能满足大部分应用的需要。在这种方式下，我们只需要研究 WEB 层的集群技术就可以了，目前在 JBOSS 平台下，一般是借用 apache+modjk 做为 web 层的集群工具，modjk 是和 apache 服务器结合对 HTTP 请求进行负载均衡的工具[17]，它的工作原理是，预先对每台服务器分配不同的权重，权重高的服务器就会负担更多的请求。这种方式虽然考虑了服务器承受能力的不同，但是，在实际应用中，各个服务器的负载能力是不断变化的，有可能权重比较高的服务器已经处理了太多的请求而变得不堪负荷。针对这种情况，有必要提出新的解方案，使用请求能够更均衡的分配到各个服务器上。

5.1 负载均衡器的基本特征

WEB层负载均衡器一般来说是位于浏览器和服务器之间，可以用硬件实现也可以通过软件实现，不管用哪种方式来实现。负载均衡器一般来说应该具有一下几个特征

(1)实现负载均衡算法

当客户端的请求到来时，负载均衡器能决定将请求转发到后台的哪个服务器实例。比较流行的算法有：轮循算法，随机算法和权重算法[18]。负载均衡器总是试图让每个服务器实例分担等同的压力。

(2)健康检查

一旦某一个服务器实例停止工作，那么负载均衡器应该能够检测到并且不再把请求转发到这个服务器实例。同样，当这个失败的服务器重新开始工作以后，负载均衡器也要能够检测到，并且开始向它发送请求。

(3)会话粘滞

几乎所有的WEB应用都会有一些会话状态。简单的可能记录了你是否登录，复杂一点的可能记录了购物车的内容等。因为HTTP协议本身是无状态的，所以会话状态就需要记录在某个地方，并且和你的浏览器相关联，以便于下次请求的时

17 S.Ranka,Y.Won and S.Sahni.Programming a Hypercube Multicomputer.IEEE Software, 1998,9:69--77

18 王玉，《揭开J2EE集群的面纱》 http://www.water-season.com/forum/viewthread.php?tid=2889

候能够很方便的取出来。当进行负载均衡的时候，对于某一个确定的会话来说，把请求转发到上一次他所请求的服务器实例是一个很好的选择，否则的话，可能会导致应用不能正常的工作。

5.2 基本的网络负载均衡算法

负载均衡算法设计的好坏直接决定了集群在负载均衡上的表现，设计不好的算法，会导致集群的负载失衡。一般的负载均衡算法主要任务是决定如何选择下一个集群节点，然后将新的服务请求转发给它。有些简单负载均衡方法可以独立使用，有些必须和其它简单或高级方法组合使用。而一个好的负载均衡算法也并不是万能的，它一般只在某些特殊的应用环境下才能发挥最大效用。因此在考察负载均衡算法的同时，也要注意算法本身的适用面，并在采取集群部署的时候根据集群自身的特点进行综合考虑，把不同的算法和技术结合起来使用。

(1)轮循法：

轮循算法是所有调度算法中最简单也最容易实现的一种方法。在一个任务队列里，队列的每个成员（节点）都具有相同的地位，轮转法简单的在这组成员中顺序轮转选择。在负载均衡环境中，均衡器将新的请求轮流发给节点队列中的下一节点，如此连续、周而复始，每个集群的节点都在相等的地位下被轮流选择。这个算法在DNS域名轮询中被广泛使用。

轮循法的活动是可预知的，每个节点被选择的机会是1/N，因此很容易计算出节点的负载分布。轮循法典型的适用于集群中所有节点的处理能力和性能均相同的情况，在实际应用中，一般将它与其他简单方法联合使用时比较有效。

(2)散列法

散列法也叫哈希法（HASH），通过单射不可逆的HASH函数，按照某种规则将网络请求发往集群节点。哈希法在其他几类平衡算法不是很有效时会显示出特别的威力。例如，在前面提到的UDP会话的情况下，由于轮转法和其他几类基于连接信息的算法，无法识别出会话的起止标记，会引起应用混乱。而采取基于数据包源地址的哈希映射可以在一定程度上解决这个问题：将具有相同源地址的数据包发给同一服务器节点，这使得基于高层会话的事务可以以适当的方式运行。相对称的是，基于目的地址的哈希调度算法可以用在Web Cache集群中，指向同一个目标站点的访问请求都被负载平衡器发送到同一个Cache服务节点上，以避免页面缺失而带来的更新Cache问题。

(3)最少连接法

在最少连接法中，均衡器纪录目前所有活跃连接，把下一个新的请求发给当前含有最少连接数的节点。这种算法针对TCP连接进行，但由于不同应用对系统资源的消耗可能差异很大，而连接数无法反映出真实的应用负载，因此在使用重型Web服务器作为集群节点服务时（例如Apache服务器），该算法在平衡负载的效

果上要打个折扣。为了减少这个不利的影响，可以对每个节点设置最大的连接数上限（通过阈值设定体现）。

(4)最低缺失法

在最低缺失法中，均衡器长期纪录到各节点的请求情况，把下个请求发给历史上处理请求最少的节点。与最少连接法不同的是，最低缺失记录过去的连接数而不是当前的连接数。

(5)最快响应法

均衡器记录自身到每一个集群节点的网络响应时间，并将下一个到达的连接请求分配给响应时间最短的节点，这种方法要求使用ICMP包或基于UDP包的专用技术来主动探测各节点。在大多数基于LAN的集群中，最快响应算法工作的并不是很好，因为LAN中的ICMP包基本上都在10ms内完成回应，体现不出节点之间的差异；如果在 WAN上进行平衡的话，响应时间对于用户就近选择服务器而言还是具有现实意义的；而且集群的拓扑越分散这种方法越能体现出效果来。这种方法是高级平衡基于拓扑结构重定向用到的主要方法。

(6)加权法

加权方法只能与其他方法合用，是它们的一个很好的补充。加权算法根据节点的优先级或当前的负载状况（即权值）来构成负载平衡的多优先级队列，队列中的每个等待处理的连接都具有相同处理等级，这样在同一个队列里可以按照前面的轮转法或者最少连接法进行均衡，而队列之间按照优先级的先后顺序进行均衡处理。在这里权值是基于各节点能力的一个估计值。

5.3 动态负载均衡器的设计

当客户访问集群资源时，提交的任务所需的时间和所要消耗的计算资源是千差万别的，它依赖于很多因素。例如：任务请求的服务类型、当前网络带宽的情况、以及当前服务器资源利用的情况等等。一些负载比较重的任务需要进行计算密集的查询、数据库访问、很长响应数据流；而负载比较轻的任务请求往往只需要读一个小文件或者进行很简单的计算[19]。对任务请求处理时间的不同可能会导致处理结点利用率的倾斜，即处理结点的负载不平衡。有可能存在这样情况，有些结点已经超负荷运行，而其他结点基本是闲置着。同时，有些结点已经忙不过来，有很长的请求队列，还不断地收到新的请求。反过来说，这会导致客户长时间的等待，而集群整体的服务质量下降。因此，有必要采用一种机制，使得平衡器能够实时地了解各个结点的负载状况，并能根据负载的变化做出调整。具体的做法是，当负载均衡器对请求进行转发时，先探测一下各结点实际的负载量，选

19tomcat主页相关内容http://tomcat.apache.org/tomcat-3.3-doc/mod_jk-howto.html

择出一个负载最低的结点，将请求转发过去，该算法考虑每一个结点的实时负载和响应能力，不断调整任务分布的比例，来避免有些结点超载时依然收到大量请求，从而提高单一集群的整体吞吐率。

各结点的负载量，主要根据各个结点的CPU 利用率、可用内存以及当前用户数状况等因素进行计算得出，各因素并不具有相同的地位，管理员可根据实际情况对各因素所占的权重进行调整。

动态负载是由结点运行时各方面的参数计算出来的。我们在实验中选取了最重要几项，包括：CPU资源，内存资源，当前用户数等信息作为计算公式的参数。对于不同类型的系统应用，各个参数的重要程度也有所不同。典型的Web应用环境下，可用内存资源和用户数量就非常重要；如果用户以长的数据库事务为主，则CPU使用率和可用内存就相对重要一些。为了方便在系统运行过程中针对不同的应用对各个参数的比例进行适当调整，我们为每一个参数设定一个常量系数Ri,用来来表示各个负载参数的重要程度，其中 ΣRi =1。因此，任何一个结点Ni 的权值公式就可以描述为：

LOAD(Ni)=R1*Lcpu(Ni)+R2*Lmemory(Ni)+R3*LSession(Ni)

其中 Lf(Ni) 表示结点 Ni 当前某一项参数的负载值，上述公式中依次表示为：CPU使用率、内存使用率、当前用户数

例如，在WEB服务器集群中，我们采用以系数{0.2, 0.4, 0.4}，这里认为服务器的内存和当前用户数较其他参数重要一些。若当前的系数 Ri 不能很好地反映应用的负载，系统管理员可以对系数不断地修正，直到找到贴近当前应用的一组系数。

另外，关于采集权值的周期设置，为了避免频繁的数据采集对应用性能的影响，采用每次转发之前对负载进行计算，而且属于同一个会话总是转发到同一台服务器上，均衡器只在初次转发时计算负载，后续的转发就不再重复计算。

5.3.1 动态负载均衡器实现环境说明

(1)J2EE服务器 Jboss4.0服务器

(2)操作系统 Windows NT 内存512M 硬盘80G

(3)开发语言 JAVA C++

(4)工具 JDK1.5,Jbuilder9,Borland c++6.0,Jmeter.

5.3.2 技术原理

(1)使用Servlet Filter截获用户请求

Java Servlet 2.3规范了过滤器[20]的功能，是管道和过滤器体系架构在J2EE

20谢夏,李胜利,金海，《基于集群服务器性能的TPC-W基准测试》，华中科技大学学报(自然科学版),2003,31(2):26-27

中的应用实践。通过使用该模式使得WEB应用开发者能在请求到达资源之前截获请求，在处理请求后修改应答。过滤器类必须j实现javax.servlet.Filter接口。这一接口包括三个方法

a)Init(FilterConfig):这是容器所调用的初始化方法。它保证了在第一次doFilter()调用前由容器调用。它能获取web.xml文件中指定的Filter初始化参数。

b)DoFilter(ServletRequest,ServletResponse,FilterChain):这是一个完成过滤行为的方法。它同样是一个过滤器调用方法。它引入的FilterChain对象提供了后续过滤器所要调用的信息。

c)Destroy()方法：容器在销毁过滤器实例的实例前，doFilter()中的所有活动都被该实例终止后，调用该方法。

(2)使用Listener监听用户会话

Listener类可以用来监听WEB应用的相关事件，包括有关生命周期和添加/删除属性的事件。Listener的生命周期由应用服务器来管理维护，而且容器在处理WEB应用请求的时候负责把触发事件传递给Listener,并且调用Listener中的相应方法。对Session的创建和注销可以通javax.servlet.http.HttpSessionlistener来监听，当HttpSession被创建或变为无效时，该事件发生。

（3）利用 JNI 技术获取平台信息。

由于 Java 具有平台无关性的特点，因此单纯的利用 JAVA 语言无法获得平台的一些信息，如内存使用情况，CPU 占用率等。我们只能采用 Java 本地接口技术（JNI）[21]来调用 C 或 C++程序来达到目的。

JNI 技术为使 Java 语言开发的应用，可以调用其他语言（例如 C 语言）所开发的应用起到了桥梁的作用。JNI 技术可以使开发人员像在 JAVA 语言中使用 JAVA 对象一样在本地方法中使用 JAVA 对象。本地方法可以创建许多 JAVA 对象，例如数组和字符串，并且可以利用这些对象执行任务。本地方法还可以检查应用由 JAVA 应用创建的对象，并且本地方法还可以更新、传递这些对象。被本地方法更新后的对象对于 JAVA 应用而言仍然是有效的。由此可见，由本地语言端（例始 C 语言）与 Java 应用代码端创建、更新的 JAVA 对象可以被互相访问，在它们之间共享。

（4）利用 XML 技术获取配置信息。

XML (eXtensible Markup Language，可扩展的标记语言)是万维网联盟(W3C)创建的一组规范，用于在Web上组织、发布各种信息。它不仅可以满足迅速增长的网络应用的需求，还能够确保网络进行交互操作时具有良好的可靠性与互操作性。它也是当前很受欢迎的数据格式，优点在于:人性化，自述性以及使用的方

[21]JNI 相关内容 http://www.itisedu.com/phrase/200604261218435.htm

便性。在本实现中，主要使用APACHE DOM4j来实现XML文件中规则的读取。

（5）服务器活动否的测试方式

本例不提供失效处理，但在转发请求前，先检测节点服务器活动与否，保证把请求只转发给活动的服务器。在此，采用的方式是通过创建url对象，并将它作为参数调用所要检测的服务器，如果返回的输入流不空，则说明服务器是活动的。

5.3.3 系统实现

（1）Filter 的实现

由于 filter 可以在请求到达所请求的资源之前截获并处理请求，所以可以利用 Filter 先获得请求后，再利用 socket 通各节点建立连接，获得各节点的负载状况，选出一个负载最小的节点将请求转发过去。Filter 类的 doFilter 方法如下：

```
public class balancer implements Filter{
public void doFilter(ServletRequest request,ServletResponse response,FilterChain chain) throws IOException,ServletException
{HttpServletRequest req=(HttpServletRequest)request;
HttpServletResponse resp= (HttpServletResponse)response;
StringBuffer requesturl=getrequesturl(req);
StringBuffer host=getliveurl();
StringBuffer responseurl=host.append("/").append(requesturl);
resp.sendRedirect(new String(responseurl));
}
}
```

Getliveurl()方法是实现的关键，它的作用是获取一个活动的且负载最低的节点地址。其内容如下：

```
String minloadurl=null,loadurl=null;
double minload=0,load=0;
minload=checkload((String)urlmap.get(0));
minloadurl=(String)urlmap.get(0);
for(int i=1;i<urlmap.size();i++)
{loadurl=(String)urlmap.get(i);
load=checkload(loadurl);
if(load<minload)
{minload=load;
minloadurl=loadurl;
```

```
}
}
return  new StringBuffer(minloadurl);
}
```

实现思路是分别和各节点建立连接，获得各节点的负载，经过比较，得出负载最小的主机地址。Checkload(String url)方法就是负责和各节点通信并获得信息的方法。

```
public double checkload(String url)
{double load=0;
parseurl(url);
try{
BufferedReader br=new BufferedReader(new InputStreamReader
(socket.getInputStream()));
String weightvalue=null;
while(!((weightvalue=br.readLine()).equals("end")))
{if(weightvalue!=null)
System.out.println("weight value is "+weightvalue);
load=new Double(weightvalue).doubleValue();
}
socket.close();
}catch(Exception e){System.out.println(e.toString());}
return load;
}
```

(2)listener 类的实现

Listener 类建立在各个服务器接点上，它的功能是，在创建和销毁 Session 时，负责对 Session 数进行记录。同时在 Listener 类初始化时，启动一个 sendmessage 线程类，这个线程的作用是建立一个服务器端的 socket,等待负载均衡器向其连接，一旦负载均衡器建立起连接，就向负载均衡器发出本机的负载状况。

```
public class loadlistener implements HttpSessionListener{
public static int activesessions=0;
public loadlistener() {
System.out.println("initial session is"+activesessions);
new sendmessage().start();
HttpSession s=null;
```

```
}
public void sessionCreated(HttpSessionEvent e)
{activesessions++;}
public void sessionDestroyed(HttpSessionEvent e)
{activesessions--;
}
public int getactivesessions()
{return activesessions;
}
}
```

Sendmessage 类的实现

```
public class sendmessage  extends Thread{
ServerSocket server=null;
Socket client=null;
BufferedWriter bw=null;
OutputStreamWriter osw=null;
OutputStream os=null;
public sendmessage() {
try{server=new ServerSocket(6666);}catch(Exception e){}
}
public void run()
{
while(true)
{
try{
client = server.accept();
BufferedReader in = new BufferedReader(new
InputStreamReader(client.getInputStream()));
PrintWriter out = new PrintWriter(client.getOutputStream(),true);
while(true)
{
double weight = calculateweight();
out.println(weight);
out.println("end");
out.flush();
```

```
client.close();
}
}catch(Exception e){System.out.println(e.toString()+"server");}
}
}
public double calculateweight()
{double freemem=(double)SystemInformation.getFreeMem();
 double cpupercent=SystemInformation.getProcessCPUPercentage();
 double weight=0.4*cpupercent+0.4*freemem+0.2*loadlistener.activeses+
 ions;
 return weight;
}
}
```

calculateweight()方法就是计算出本机的负载。计算本机空闲内存和 CPU 使用率的方法分别是 SystemInformation.getFreeMem() 方法和 SystemInformatiion.getProcessCPUPercentage()方法.

SystemInformation 类是一个抽象类，它的功能是通过调用 C 语言编写的本地函数获取系统的相关信息。getFreeMem()的函数原型是

```
public static native long   getFreeMem ();
public static native double getProcessCPUUsage ();
JNIEXPORT jlong JNICALL
```

getFreeMem()的 C 语言实现如下：

```
Java_com_vladium_utils_SystemInformation_getFreeMem (JNIEnv * env,
jclass cls)
    {
     MEMORYSTATUS stat;
     GlobalMemoryStatus (&stat);
      return (jlong)(stat.dwAvailPhys/1024);
}
```

通过WINDOWS系统进程状态支持模块PSAPI.dll[22]中的GlobalMemoryStatus()函数获取可用内存信息。

getProcessCPUUsage ()的 C 语言实现如下：

```
JNIEXPORT jdouble JNICALL
```

22朱利，张兴军，《Web服务器组的负载均衡方法研究》，小型微型计算机系统.2003 24(12)

```
    Java_com_vladium_utils_SystemInformation_getProcessCPUUsage
    (JNIEnv * env, jclass cls)
{
    FILETIME creationTime, exitTime, kernelTime, userTime, nowTime;
    LONGLONG elapsedTime;
    DWORD errCode;
    LPVOID lpMsgBuf;
    BOOL resultSuccessful = GetProcessTimes (s_currentProcess, &
    creationTime, & exitTime, & kernelTime, & userTime);
    if (!resultSuccessful) {
     errCode = GetLastError();
     FormatMessage(
     FORMAT_MESSAGE_ALLOCATE_BUFFER |
     FORMAT_MESSAGE_FROM_SYSTEM,
     NULL,
     errCode,
     MAKELANGID(LANG_NEUTRAL, SUBLANG_DEFAULT),
     (LPTSTR) &lpMsgBuf,
      0, NULL );
      printf("[CPUmon] An error occured while trying to get CPU time.
      Error code: %ld  Error description: %s",errCode,lpMsgBuf);
       fflush(stdout);
       LocalFree(lpMsgBuf);
       return -1.0;
      }
    GetSystemTimeAsFileTime (& nowTime);
    elapsedTime = fileTimeToInt64 (& nowTime) - fileTimeToInt64 (&
    creationTime);
    if (elapsedTime < MIN_ELAPSED_TIME)
    return 0.0;
    else
    return ((jdouble) (fileTimeToInt64 (& kernelTime) + fileTimeToInt64
    (&userTime)))/(s_numberOfProcessors * elapsedTime);
}
```

5.4 测试用例及结果比较

测试用例：用户登录的 JSP 调用。将其所在的应用部署到各群集服务器实例，对于同样的应用，同样的服务器环境，分别比较 APACHE+MOD_JK2+JBOSS 的集群方式和上例所实现动态负载均衡的方式。集群由三台服务器组成，另专有一台服务器做为负载均衡器。测试工具使用 JMETER[23]。测试结果如下：

(1)APACHE＋MOD_JK2 方式的集群

在测试 APACHE＋MOD_JK2 集群时，在服务器二和服务器三上运行一些应用程序，故意耗费一些内存和 CPU 资源。

表 5.1 使用 APACHE＋MOD_JK2 实现的集群性能测试结果表

测试频率(次数/秒)	持续时间	测试次数	成功次数	成功率	平均响应时间（毫秒）
5	7	35	35	100%	14
4	7	28	28	100%	12

资料来源：本研究整理

表 5.2 各服务器上分配的请求数目及 CPU 及内存使用率

服务器	请求数目	CPU 占用率	内存使用率
服务器一	21	37．25%	33%
服务器二	22	55%	65%
服务器三	20	74%	92%

资料来源：本研究整理

（2）带有负载检测的集群

表 5.3 使用带有负载检测的集群性能测试结果表

测试频率(次数/秒)	持续时间（秒）	测试次数	成功次数	成功率	平均响应时间（毫秒）
5	7	35	35	100%	15
4	7	28	28	100%	14

资料来源：本研究整理

23Gorton,L;Liu,A;Brebner,P.Rigorous evaluation of COTS middleware technology Computer,Volume:36 Issues:3,2003,3:50-55

表 5.4 各服务器上分配的请求数目及 CPU 及内存使用率

服务器	请求数目	CPU 占用率	内存使用率
服务器一	27	37. 25%	33%
服务器二	21	55%	65%
服务器三	15	74%	92%

资料来源：本研究整理

结果比较：

方案（1）的优点在于借助现成的集群模块，不用针对集群进行开发，减少了工作量。可以把请求均匀的转发到各个服务器上，而且速度要略快于方案(2)。缺点是，没有考虑各个服务器上的实际负载情况，只是简单的对请求进行平均分配。

方案（2）的优点在于在分配请求时，考虑到了服务器的实际负载能力，给当前负载较低的服务器分配了更多的任务，缺点是，由于要对各服务器的负载进行计算，转发速度要略慢于方案（1）。

5.5 本章小结

本章提出了基于负载检测的负载均衡方案，并予以实现，最后针对 JBOSS 平台采用了两种不同方式的 WEB 集群的效果进行了测试并比较结果。

第六章 废旧家电回收信息管理系统的优化实践

自 2004 年起，欧盟各国陆续通过了欧盟环保法（WEEE）,该法规定，家电生产企业必须对其在欧洲销售的家用电器进行回收，如无法回收，家电生产企业必须向当地政府缴纳一部分的回收处理费用。我国是家用电器出口大国，此法律的制定，对我国的家电出口行业带来较大的影响，为了能够及时准确对我国出口的家电产品进行回收处理。中国家用电器研究院受法制办的委托进行废旧家电回收信息管理系统的研发工作。笔者有幸参与了此项目的开发，此项目如果最终能够投入运行的话，将会是一个庞大的系统，要涉及家电产品生产、销售、出口、回收的各个环节，因此，在前期的初步试验阶段，必须十分仔细的对系统性能进行优化，以便于日后系统能高效、稳定的运行，并且具有可扩充的能力。

笔者在开发过程中，综合利用了前几章中研究过的优化技术，消除了系统中的内存泄露，设置了性能较为优越的实例池，并将动态负载均衡技术应用该项目。通过这些技术的应用，使该系统的性能及稳定性得到了较大幅度的提升。

6.1 系统介绍

废旧家电回收信息管理系统，是基于 J2EE 平台的三层 WEB 系统，系统的作用是利用 RFID 技术对家电产品的回收信息进行采集，便于有效的对回收的旧家电进行处理和统计。

6.1.1 系统体系结构

该系统采用 J2EE 和 JBOSS 建立起来基于 Web 的三层技术架构，该系统设计利用了 JBOSS 服务器的性能，可升级性等能力。该系统有两个逻辑层面：表示层和模型层。如下图所示，客户机使用 WEB 浏览器与表示层通信，表示层负责将客户的需求传递到业务逻辑层，并将业务逻辑层的响应传递到客户端。

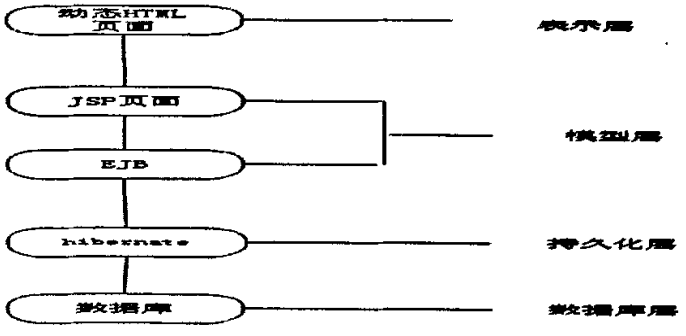

图 6.1 系统结构示意图

资料来源：本研究整理

业务逻辑层处理应用程序逻辑和通信，例如，与数据库这样的后端系统的通信。该数据库为应用程序数据提供永久储存的仓库，通常置于单独的服务器上。表示层用包括 JSP 页面和标记符库的 WEB 组件来实现。该系统用 EJB 组件和 Hibernate 实现了业务逻辑层。

6.1.2 系统功能模块设计

系统按照系统－分系统－子系统－模块的组织形式进行划分。系统分为 CUID 管理中心系统，家电生产企业系统，家电零售企业系统,家电回收处理系统四个分系统。系统结构如下图：

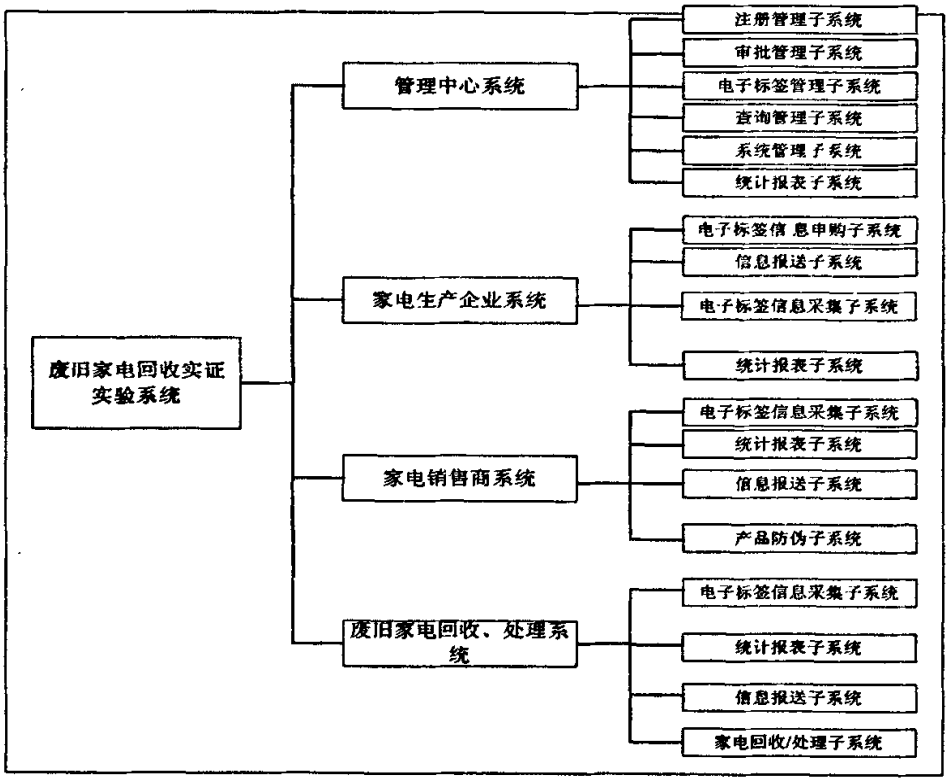

图 6.2　系统功能模块图

资料来源：本研究整理

6.2 系统内存泄漏问题的检测与解决

在对系统进行测试的过程当中，发现在长时间的运行后，响应速度会变得很慢，通过系统管理器查看，发现服务器的可用内存资源变得很少。我们初步认定这种情况很可能是系统发生了内存泄漏问题。

为了研究这个问题，我用 JProfiler 来确定什么地方出了差错。Jprofiler[24]是一

24 Yan Liu, J. Performance and scalability measurement of COTS EJB technology. Computer Architecture and High Performance Computing, 2002, 10:212-219

个优秀的 JVM 剖析工具。首先，要证实是否的确存在内存泄漏的发生。

JProfiler 提供了一个称为 Heap Walker 的视图，它显示 Java 应用程序运行时所占用的堆内存量随时间的变化。它还提供了一个工具栏按钮，必要时可以强制 JVM 执行垃圾收集。如果您试图弄清楚，当 Java 应用程序不再需要给定的类实例时，这个实例会不会被作为垃圾收集，这个功能将很有用。图 6.3 显示了使用中的堆存储量随时间的变化。

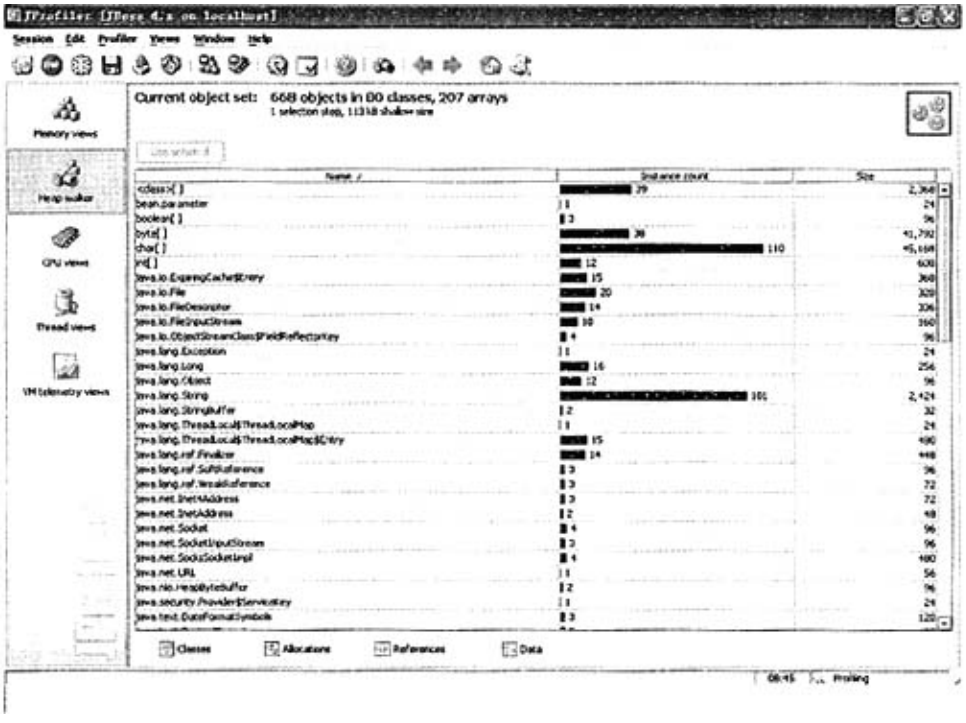

图 6.3 Heap Walker

资料来源：本研究整理

在 Heap Walker[25]视图中，红色部分表明已分配的堆空间大小。在启动这个 Java 程序并达到稳定状态以后，我强制垃圾收集器运行，随后，我多次重复登录该系统，并再次调用了垃圾收集器。这时我通过查看 Instance Summary 证实确实有一个漏洞，因为 Instance Summary 表明 bean.parameter 类的计数在检查点之后增加了 4 个。

确定的确发生内存泄漏后，下一步的工作就是要查明使垃圾收集器无法正常完成其工作的那些引用，我使用 JProfiler 的 Reference Graph（如图 6.4 所示）来确定哪些类仍然引用着目前未被删除的 bean.parameter 类。在调试这个问题时

25 C. Z. Xu and F. C. M. Lau. Analysis of the Generalized Dimension Exchange Method for Dynamic Load Balancing. Journal of Parallel Distributed Computing, 1992, 16(4)

该过程是最复杂的过程之一，因为我发现许多不同的对象仍然引用着这个无用的对象。用来查明究竟是哪个引用者真正造成这个问题的试错过程相当耗时。在本例中，一个根类是问题的发源地。右侧显示的类处在从最初的 bean.parameter 类跟踪而来的路径上。

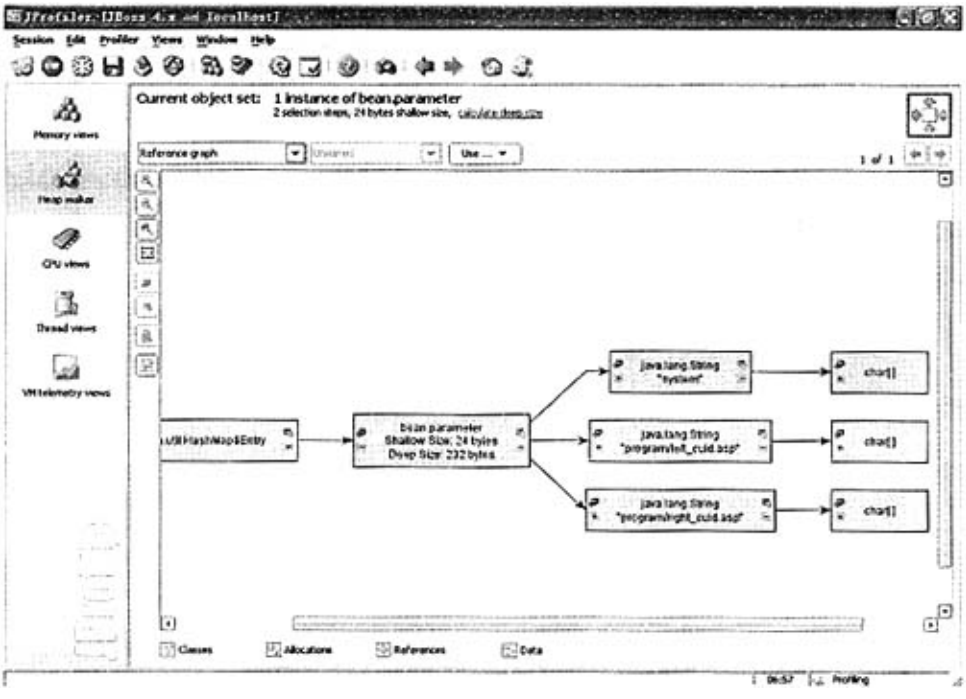

图 6.4. 在引用图中跟踪内存漏洞

资料来源：本研究整理

就本例而言，最后查明罪魁祸首是包含一个静态 hashmap 的登录管理器类。通过逆向追踪引用者列表，我发现根节点是用来存储每个用户登录信息的静态的 map 类实例当中。

这个用户信息管理器类的问题是，它是一个全局的单实例，其存放的内容永远不会释放，其存放的内容是用户的登录标识和一个表明该用户拥有权限的菜单的 ArrayList 类组成，用户登录标识号由用户 ID 加上登录时间戳组成。每天用户登录所形成的登录标识都不一样，就造成了每天的登录信息（包括菜单信息）都保存在内存中不会被删除，从而造成了内存泄露。查到内存泄漏的原因后，修改变得很简单了，只需要采用任务定时清理每天的过期会话登录信息或线程轮询清理即可。

当作了这些修改以后，再次运行系统，系统表现稳定，不再出现响应速度变慢的问题。

6.3 动态负载均衡器在废旧家电回收信息管理系统上的应用

为了提高系统处理用户请求的能力，增强系统的稳定性和可用性，系统使用了集群技术。集群是由一台负载均衡器和三台服务器组成。负载均衡器是由部署了第五章中笔者设计的动态负载均衡器后的一台服务器充当。在集群系统搭建完毕后，开始对集群性能的测试工作。

6.3.1 加压和性能测试方法

为了检验该系统在集群环境下的性能，需要对该系统进行加压和性能测试。当有许多客户机同时连接到用户网站时，集群配置可以自动测试各结点的负载，并将请求转发当前负载较轻的结点上去。

为了检验集群的这种性能，加压和性能测试尝试仿制将在该应用程序上出现的负载。

JBOSS 应用程序的加压和性能测试方法按以下 4 个步骤进行：

(1) 定义测试范围：

a) 80%的交互作用将是对数据库进行只读访问，如查询或统计数据，生成数据报表，这是日常工作中最频繁的操作。

b) 20%的交互作用将包括数据库的更新。

(2) 设计测试：根据前面的测试范围设计测试作用模型。

a) 模仿许多用户同时查询数据。

b) 模仿许多用户同时添加数据。

c) 模仿许多用户 80%读信息，20%写信息。

(3) 实现和执行测试，检验应用服务器和数据库服务器的效率，如 CPU 利用率、内存利用率等。

(4) 审查结果和重复测试，保证测试结果的可靠性。

6.3.2 测试环境

(1) 软硬件环境配置

a) 数据库服务器：P4 2.4G/1G/IDE/100M PCI 网卡/Oracle9i。

b) 应用服务器：P4 2.4G/1G/IDE/100M PCI 网卡/JBOSS/废旧家电回收信息管理系统服务器端程序。

c) 客户机：P4 2.4G/512M/IDE/100M 网卡/Windows XP。

(2) 网络配置：3COM 3900 Fast Ethernet Switch 10M/100M 自适应快速以太网交换机。

(3) 为了模拟客户访问，使用了负载测试工具 Jmeter 它是一个全功能的 Web 应用程序负载产生工具，其中具有通过 Web 浏览器记录用户交互作用的能力。此外，它还能够模仿许多不同的用户，每个用户都用不同的用户名和密码登陆来进行模拟测试。系统采用这种方式来模拟测试集群负载均衡的性能。

6.3.3 测试结果

在每个客户端机器上使用 Jmeter 重复访问单个 URL 的功能。以便模拟大量的用户进行压力测试。同时，可以通过查看 Win2K 处理器监视器，显示应用服务器和数据库服务器的 CPU 利用率和内存利用率，部分性能参数计算公式如下：

(1)吞吐量=负载程序数/总耗费时间=负载程序数/(负载程序数*平均查询时间*0.8十负载程序数*平均写入时间*0.2)

(2)平均执行时间=平均查询时间*0.8十平均写入时间*0.2

如下表所示的正常运行情况(80%读，20%写)下的检测结果。

表 6.1 使用 APACHE+MODJK 方式集群

负载请求数	平均写入时间（秒）	平均查询时间（秒）	吞吐量	平均执行时间	应用服务器 1		应用服务器 2		应用服务器 3	
					Cpu 使用率	内存使用率	cpu 使用率	内存使用率	cpu 使用率	内存使用率
150	6.2	4.9	0	5.34	18.90%	20.80%	26.50%	17.30%	72%	17.20%
200	5.4	4.7	0	5.18	30.20%	22.40%	29.30%	19.50%	6%	29.40%

资料来源：本研究整理

表 6.2 采用动态负载均器进行集群

负载请求数	平均写入时间（秒）	平均查询时间（秒）	吞吐量	平均执行时间	应用服务器 1		应用服务器 2		应用服务器 3	
					cpu 使用率	内存使用率	cpu 使用率	内存使用率	cpu 使用率	内存使用率
150	6.2	4.9	0	5.34	18.90%	20.80%	26.50%	17.30%	72%	17.20%
200	5.4	4.7	0	5.18	30.20%	22.40%	29.30%	19.50%	6%	29.40%

资料来源：本研究整理

根据测试结果可以看出，随着负载增加，如集群服务器负载请求数从 150 上升到 200 时，采用轮询法的集群，在高负载情况下，由于负载集中导致集群内部分节点资源空闲、部分节点资源分布不均的情况，影响了集群的效率，从测试可以看出，当集群负载从 150 上升到 200 时，集群的负载集中在应用服务器 2 中，其平均执行时间从 5.22 上升到 6.42，平均执行时间增加了 1.2 分钟;采用基于动态负载检测的均衡的集群系统能够随着负载增加而合理分配负载到集群内的节点中，理论计算的平均执行时间从 4.48 上升到 4.98，平均执行时间只增加了 0.4 分钟。

采用轮循法的负载均衡策略的集群系统，在高负载情况下，根据集群应用服务器的实现方式会有不同的结果，从测试中可以看出，应用服务器 2 的负载与应用服务器 1 的负载分布不均匀，从而造成集群资源没有合理利用，照此情况推断，当集群负载达到一定情况下，集群中的某台应用服务器将会出现负载过大从而宕

机的可能，这不仅影响集群的性能、同时也影响集群的可靠性，而基于动态反馈的负载均衡算法，引入了节点负载增量的概念，并对节点实际负载进行预测。同时，通过动态反馈方法进行修正，有效地实现了负载均衡，保证了负载均衡集群在长时间的运行过程中不会发生大的倾斜，处理能力强的节点获得任务的概率相对较高，提高了集群系统的整体效率。

6.4 本章小结

本章简单介绍了笔者参与开发的废旧家电回收管理信息系统的情况，并将前几章所研究的几种优化技术应用到该项目的实践中来，并取得了一定的效果。

第七章 结论

7.1 总结

本文深入分析了 J2EE 平台的基本理论与体系架构。并在此基础上分析了影响 J2EE 应用性能的若干因素及提升 J2EE 应用性能的种种方案。本文主要在以下方面进行了研究：

（1）研究 J2EE 平台应用服务的技术特征，从体系结构上对其进行分析，着重对 EJB 相关组件技术在 J2EE 平台中的作用和部署情况进行详述。

（2）对于 JAVA 的内存管理机制进行深入剖析，重点研究了内存泄漏的问题。

（3）研究 EJB 技术及其在 JBOSS 平台下的优化。

（4）深入地分析了J2EE平台各类组件的特点，重点阐述了各类集群服务的负载均衡的解决方法和相关技术。

（5）对于WEB层集群的负载均衡技术进行研究，并提出了按照服务器负载进行请求转发的方案并予以实现。

在本研究中，创新点主要体现在两点：

（1）对于内存泄漏的检测问题提出了自己的解决方案。

（2）对于Servlet容器层的负载均衡提出了基于服务器负载动态检测的实现方式。这种实现方式考虑到了各服务器实时的负载情况变化，有利于更均匀地在集群内部分配任务。

7.2 后续工作

由于研究水平和研究时间的限制，本课题还有许多有待完善或值得继续探讨的理论和应有问题。

（1）继续完善内存泄漏的检查工具，以一种更直观，更智能的方式将检查结果呈现给使用者。

（2）从集群的角度而言，当集群中的一个节点失效时，集群能够自动将对该节点的请求转发到集群中另外一个正确运行的节点上，保证集群的可靠性，这个功能称为失败转发(Fail-Over)，如何实现失败转发是一个有意义的研究方向。

（3）本文只实现了WEB层的负载均衡，虽然在WEB层进行负载均衡是一个性能比较理想的方案。但是对于一些特殊的应用来说，在EJB层进行集群是必须的，因此，有必要进一步深入研究EJB层负载均衡的机制并予以实现。

参考文献

[1]谢小乐，《J2EE经典实例详解》，人民邮电出版社 2002,235-239

[2]斯瑞格奈斯，《精通EJB》，电子工业出版社 2006,56-59

[3]杨绍方，《深入掌握J2EE技术》，科学出版社 2002,192-198

[4]王森，《JAVA深度历险》，麦格罗.希尔国际出版社,2001,102-104

[5]林胜利，王坤茹，孟海利，《JAVA优化编程》，电子工业出版社，2002,101-107

[6]bill venners，《深入JAVA虚拟机》，机械工业出版社 2003,143-145

[7]吉庆洋，《JAVA内存泄露》

http://www.blogchinese.com/06081/242496/archives/2006/200691118517.shtml

[8]约翰逊，《J2EE设计开发编程指南》，清华大学出版社 2001,156-160

[9]Martin bond,Dan Haywood,Debbie Law and Peter Roxburgh "Teach yourself in 21 days" Sams ,2002.

[10]Sun Micro systems, Java2 platform Enterprise Edition1, 3 Specfication, Public draft,2000:22-57

[11]Altendorf,EHohman,Zabicki,Using J2EE on a large Web-based project,Software, IEEE, Volume,2 002,3-5:81-89

[12]肖辉，《应用服务器中间件EJB容器群集机制研究》，西安交大硕士论文，2003:8-9

[13]陈耿眠，晏蒲柳，郭成城等，《基于Web内容的集群服务器请求分配系统的研究与实现》，计算机工程，2003(01):87-92

[14]骆宗阳，董玮文，杨宇航等，《一种具有高可用性的通用负载均衡技术》.计算机工程，2003(02):23-27

[15]P.Kruegerand,M, Livny.A comparison of Preemptivge and Non-Preemptive Load Distributing, Technical Report,Department of Computer Science,University of Wisconsin ,Madison,WI,1998:58-96

[16]M.H.Mickle and J.M.Paul.Load Balancing Using Heterogeneous Processors for Continuum Problem on a Mesh.Journal of paralle land Distributed Computing,1996:66-73

[17]S.Ranka,Y.Won and S.Sahni.Programming a Hypercube Multicomputer.IEEE Software,1998,9:69—77

[18]王玉，《揭开J2EE集群的面纱》，

http://www.water-season.com/forum/viewthread.php?tid=2889

[19]tomcat主页相关内容

http://tomcat.apache.org/tomcat-3.3-doc/mod_jk-howto.html

[20]谢夏,李胜利,金海，《基于集群服务器性能的TPC-W基准测试》，华中科技大学学报(自然科学版),2003,31(2):26-27

[21]JNI相关内容 http://www.itisedu.com/phrase/200604261218435.html

[22]朱利，张兴军，《Web服务器组的负载均衡方法研究》，小型微型计算机系统.2003 24(12)

[23]Gorton,L;Liu,A;Brebner,P.Rigorous evaluation of COTS middleware technology Computer,Volume:36 Issues:3,2003,3:50-55

[24]Yan Liu,J.Performance and scalability measurement of COTS EJB technology. Computer Architecture and High Performance Computing,2002,10:212-219

[25]C.Z.Xu and F.C.M.Lau.Analysis of the Generalized Dimension Exchange Method for Dynamic Load Balancing.Journal of Parallel Distributed Computing,1992,16

致 谢

在我的毕业论文即将完成之际，在我即将完成硕士研究生学业之时，我要衷心地感谢两年来指导过我的老师们和热心帮助过我的同学和朋友们。

本论文的完成，得益于我的导师周志忠老师的悉心指导和无倦的教诲。导师的鼓励和帮助增强了我的自信心，使我有机会学习当前最先进的计算机技术。导师的正确指导，使我少走了不少弯路。导师务实、灵活、严谨的工作作风和治学态度，让我受益良多，这对我今后的生活和工作都有重要的指导作用。在日常的工作中，导师的信任使我得到了充分的发展和锻炼，在实践过程中去学习、探索。导师还对论文提出了很多宝贵的建设性意见。在本论文完成之际，向导师致以最诚挚的敬意和谢意。

感谢陈恭和老师，感谢他在我学习和工作中给我的关心和帮助。陈老师严谨的治学态度、宽厚的待人作风使我受益匪浅，在此对陈老师表示衷心的谢意。

最后，感谢我生活过并永远热爱的地方—对外经贸大学！