# 实体 Bean 容器管理持久化研究与实现*)

胡建华[1,2] 宦荷卿[1,2] 陈宁江[1,2]

(中国科学院软件研究所 软件工程技术研发中心 北京100080)[1]

(中国科学院软件研究所 计算机科学重点实验室 北京100080)[2]

**摘 要** 实体 Bean 是一种重要的 J2EE 应用组件,具有持久保存状态的特性。实体 Bean 容器是实体 Bean 的运行环境,持久化管理器是其中具体负责持久化的组件。本文研究了以关系数据库作为存储介质实现容器管理持久化的理论,提出了一种灵活高效的实现方法。该持久化管理器通过静态映射模型创建持久化方案,并创建动态模型以响应客户请求。在实体 Bean 容器并发访问控制机制的帮助下,该持久化管理器能够保证实体 Bean 高效可靠的运行。该持久化管理器和容器已经在中科院软件所自主研制的 J2EE 应用服务器 OnceAS 中得到实现。

**关键词** 实体 Bean,容器管理持久化,实体 Bean 容器,J2EE 应用服务器

## Research and Implementation of the Container Managed Persistence of Entity Bean

HU jian-Hua[1,2] GUAN He-Qing[1,2] CHEN Ning-Jiang[1,2]

(Technology Center of Software Engineering, Institute of Software, Chinese Academy of Sciences, Beijing 100080)[1]

(Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing 100080)[2]

**Abstract** Entity Bean, which is able to have its state persisted, is an important type of J2EE application component. Entity Bean Container provides the runtime environment for Entity Bean. As a component of Entity Bean Container, the persistence manager takes the responsibility of persisting. This paper studies how to construct a persistence manager with relational database, and presents an implementation. The implementation provides a static mapping model to construct the persistence schema, and a dynamic model to response to requests of the client. When more than one user accesses the same Entity Bean instance concurrently, the persistence manger may not work properly. To solve the problem, the container provides a mechanism to synchronize the concurrent accesses, which is also described in this paper. The persistence manager and container have been implemented in OnceAS, which is an application server compliant with J2EE1.3 specification.

**Keywords** Entity Bean, Container managed persistence, Entity container, J2EE application server

## 1 引言

J2EE 应用服务器是遵守 J2EE 规范[1]的网络应用运行支持平台。根据 J2EE 规范,J2EE 应用服务器必须支持使用 Servlet、JSP 以及 EJB 等技术编写的应用组件。其中 EJB 是一种服务器端组件体系结构,通过 EJB 能够开发出可扩展的、健壮的和安全的网络分布式应用程序。

EJB 规范定义了三种不同功用的标准 Bean:会话 Bean、实体 Bean 和消息驱动 Bean。这三种 EJB 在编写、使用、运行支持等方面各有特点[2]。本文讲述的是对实体 Bean 的支持。实体 Bean 是数据和业务逻辑的结合体,数据持久保存在企业信息系统中,实体 Bean 的实例是数据实体的代表。实体 Bean 的持久化机制实现实体 Bean 实例状态的持久化保存。根据 EJB2.0规范,有两种方式实现实体 Bean 的持久化:Bean 管理的持久化(BMP)和容器管理的持久化(CMP)[2]。对 BMP 的支持比较简单,不是本文的重点内容。对 CMP 的支持比较复杂。本文提出了使用对象-关系映射的方法实现 CMP,这种方法将实体 Bean 映射的静态模型映射到数据库中的表,将动态模型映射到内存对象,并利用应用服务器的一些相关服务实现静态模型和动态模型的结合。实体 Bean 容器是实体 Bean 的运行环境,保证实体 Bean 的事务性、安全性和可靠性,并且通过并发控制机制保证多用户访问的安全性。

本文所描述的持久化机制不仅严格遵守 EJB2.0规范的实体 Bean 规范,还提供了提高运行效率和可靠性的机制;同时实体 Bean 容器提供良好的并发控制机制,确保实体 Bean 正确运行。本文的工作在中科院软件所自主开发的 J2EE 应用服务器 OnceAS 中已经实现。

本文第2节介绍实体 bean 容器的整体结构;第3节给出实体 Bean 的静态映射模型;第4节阐述了实体 Bean 的动态映射模型;第5节对实体 Bean 容器的并发访问的控制机制进行了研究;最后是本文的小结。

## 2 支持持久化管理的实体 Bean 容器

实体 Bean 容器是实体 Bean 运行的环境。每个已部署的实体 Bean 类型与一个实体 Bean 容器的实例一一对应。根据实体 Bean 部署描述信息,对应的容器实例创建 Interceptor 链,提供事务、安全、持久化等支持[3]。实体 Bean 容器通过 JCA 连接到外部的数据库,持久化机制使用数据库实现实体 Bean 实例的持久化保存。Container-Invoker 是客户端访问实体 Bean 对应容器实例的接口。

*)基金项目:973网构软件中间件平台模型和框架研究(2002CB312005);863网络环境的系统软件核心技术及运行平台(2001AA113010);863面向新型 ERP 的可重配 Web 应用服务器研究与应用(2003AA413010)。胡建华 硕士研究生,主要研究领域为网络分布式计算;宦荷卿 博士研究生,主要研究领域为网络分布式计算;陈宁江 博士研究生,主要研究领域为网络分布式计算。

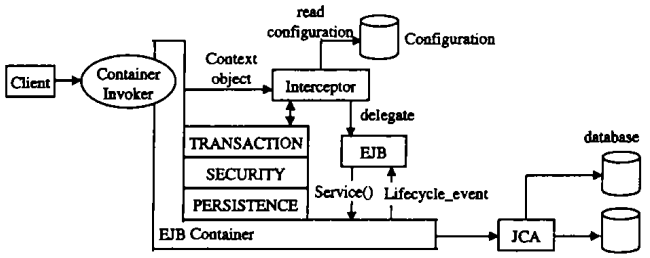


图1 实体 Bean 容器结构图

持久化管理器解析部署描述文件，确定 Bean 实例的数据库保存方案。由于是为了保存实体 Bean 的静态属性值，本文将这一映射称为实体 Bean 的静态映射。实体 Bean 的实例在内存中运行，但是实体 Bean 的运行状态保存在关系数据库中，实体 Bean 的持久化机制透明地实现内存实例与底层数据库之间的同步。为此，持久化管理器根据部署描述信息和实体 Bean 类文件生成内存模型，本文称之为实体 Bean 的动态映射。

## 3 实体 Bean 的静态映射

实体 Bean 的静态映射是为了保存一个或者多个实体 Bean 实例的状态。单个实体 Bean 的持久化方案是正确保存实例状态的基础；此外它还需要将实体 Bean 之间的关系进行映射，才能保证实体 Bean 基于对象的特性。

### 3.1 单个实体 Bean 的映射

EJB 规范规定实体 Bean 的部署描述信息必须包括持久化域的名称及其 Java 类型；实体 Bean 必须定义主键类，该类的属性必须是持久化属性的子集，这些属性称为主键属性。两个实体 Bean 如果属于同一类型，并且主键属性分别相等，则被认为是相等的。持久化管理器在数据库中创建存储方案需要确定三个要素：表名与列名，各列的数据类型，主键和外主键约束。其中表名和列名都通过部署描述信息提供；列的数据类型通过 Java 类型映射到数据库的类型，不同的数据库有不同的映射；主键约束通过部署信息的主键属性确定，外主键约束根据参与的关系确定，不属于单个实体 Bean 映射的范围，在下文讨论。

### 3.2 继承关系的处理

将继承关系映射到关系数据库中的表有多种实现方法，比较典型的有具体类映射法、组合法、直接映射法[4]。它们的区别在于如何组织子类与父类的属性。具体类映射法将父类的属性加入到子类的属性列表中，每个类都对应独立的表；组合法将所有类映射到一个表中，通过一个标志位标志某个元组是某个类的实例；直接映射法通过各个类分别建表，通过外主键关联。本文选择直接映射法，以图2(a)所示的类关系为例，映射的结果如图2(b)所示。

这种映射方法将每个类映射到数据库中的一个表，该表的属性包括一个附加的 OID 和专属于该类的属性，其中 OID 是对象在内存中的 ID。由于任何一个 Student 或者 Professor 的对象必然也是一个 Person 对象，因此 personOID 不仅是三者的主键，还是 Student 和 Professor 相关于 Person 的外主键。

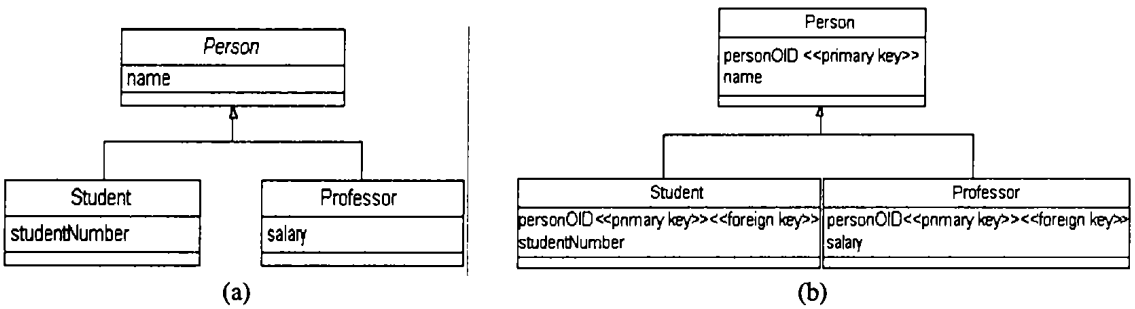


图2 实体 Bean 继承关系的 OR 映射

使用这种映射方式会增加表的数目，同时读写数据需要更复杂的关系操作，从而需要较长的执行时间。但在实际应用中，实体 Bean 对应的数据库比较小，修改比较容易；关系数据库对查询的优化策略已经比较成熟，复杂查询对速度的影响并不明显。而且这种映射方式严格遵守对象编程的思想，灵活性较高：如果修改基类，只需要修改基类对应的表；如果增加子类，则只需要增加对应类的表。

### 3.3 关联关系的 OR 映射

在对象编程中，关联关系和组合/聚合关系是常见关系。存在组合/聚合关系的对象，两者是整体与部分的关系；存在关联关系的对象之间比较独立。在图3中 Wing 与 Airplane 之间是组合/聚合关系，Airport 和 Airplane 之间是关联关系。本节讲述关联关系的 OR 映射，组合/聚合关系的 OR 映射在下一节讲述。

关联关系可以分为四种类型："单对单"、"单对多"、"多对多"和"多对单"。对于"单对多"的对象之间的关联关系，可以通过在关系变量上增加外主键约束实现。外主键约束是指对任意两个关系变量 $A$ 和 $B$，$A$ 的码 $(A_1\#, A_2\# \cdots A_n\#)$，$B$ 的元组的特定属性组 $(B_1, B_2 \cdots B_n)$ 与 $A$ 的码一一对应，$B$ 在这些元组的投影必须是 $A$ 在码上的投影的子集，这一约束称为外主键约束，$(B_1, B_2 \cdots B_n)$ 称为 $B$ 关于 $A$ 的外主键。使用外主键约束实现"单对多"的关系具体分为两个步骤：确定"单"方的主键属性；在"多"方创建保存根据"单"方的主键属性。运行时，"多"方的实体 Bean 实例保存与它相关联的实例的主键属性。

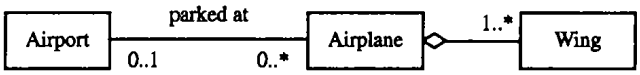


图3 关联关系与组合/聚合关系之间的区别

例如图3中的 Airport 和 Airplane，本文选择外主键约束来实现关系的 OR 映射。以 Airport-Airplane 为例，假设有以下的 Airport 元组：

airport("20040706","Beijing")

如果要创建一个 Airplane 对象与 airport 关联，映射模型将产生如下的 airplane 元组：

airplane(123,"bk","20040706")

虽然以"单对多"为例,但使用外主键约束同样可以实现"单对单"和"多对单"的关联关系。"多对多"的关联虽然不能直接用外主键约束实现,但可以按照如图4所示的方法转换成一个"单对多"和一个"多对单"关系的组合。其中 Associative Table 以 $A$ 的码和 $B$ 的码组成复合码,并保持与 $A$、$B$ 的外主键约束。

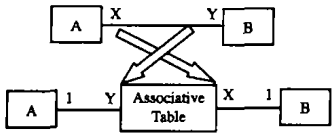


图4 "多对多"关联关系的 OR 映射

### 3.4 聚合关系的 OR 映射

再考虑图3所示的情形,Airplane 与 Wing 之间存在组合/聚合的关系,Airplane 是整体类,而 Wing 是部分类。有多种方式实现这一关系的映射,比较典型的有关系转换法、组合表法和序列化保存法。

关系转换法认为参与关系的两个类是相互独立的,从而将组合关系转换为"单对单"或者"单对多"的关联关系。例如,可以认为 Airplane 与 Wing 之间是"单对多"的关联关系。经过这样的转换,就利用外主键约束实现其 OR 映射。

组合表法认为参与关系的两个类是一个整体,因此使用一个关系变量来存放其属性,此时关系变量的属性是整体类属性与部分类属性的并集。例如 Airplane 与 Wing 分别有如下表示:

Airplane(attr1,attr2,attr3,…,attrn)

Wing(attr1,…,attrm)

并且每个 Airplane 对象与两个 Wing 对象有组合聚合关系,那么该组合映射的关系变量如下所示:

Airplane-two-wing (a-attr1, …, a-attrn, w1-attr1, …, w1-attrm,w2-attr1,…w2-attrm)。

序列化方法也认为参与关系的两个类是一个整体,更进一步,认为部分类是一个属性,因此将部分类序列化后保存为一个字节流,读取时再恢复为部分类对象。同样使用上述的 Airplane 与 Wing,使用该方法映射的结果如下:

Airplane-two-wing(a-attr1,…,a-attrn,w1-ser,w2-ser)

使用关系转换法具有较高的灵活性,但不符合 OOP 中组合/聚合关系的特点。本文采用后两种映射方式相结合的方法。根据 EJB 规范,如果部分类符合 Javabean 的编程规范则使用组合类的映射方式,否则使用序列化保存的映射方式[2]。

## 4 实体 Bean 动态模型

### 4.1 模型概述

实体 Bean 的动态模型处理客户请求,处理方法包括调用实体 Bean 自身的业务逻辑,生成数据库操作命令并提交给数据库执行。通过解析实体 Bean 部署描述文件,持久化管理器获得如下信息:需要持久化的属性及其 Java 类型,实体 Bean 业务逻辑的实现代码,实体 Bean 的查询语句,实体 Bean 之间的关系及其映射方法。通过这些信息,持久化管理器可以将实体 Bean 分解成若干个域、方法、关系角色的组合[5]。该动态模型的主要部件如图5所示。

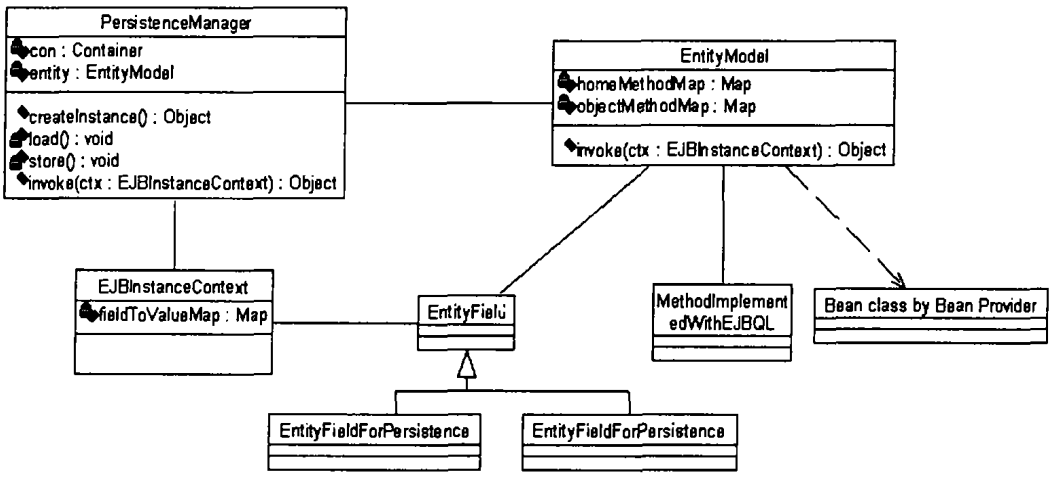


图5 实体 Bean 动态模型的实现类图

1)EntityModel 是模型的枢纽,通过它可以获得持久化属性,相关联的实体 Bean 实例,以及 EJBQL 实现的查询方法。

2)FieldBridge 代表实体 Bean 的域。实体 Bean 的域有两种:一种是实体 Bean 的持久化域,对应实体 Bean 的持久化属性,由 CMPFieldBridge 表示;另一种是关系域 CMRFieldBridge,每个关系域对应实体 Bean 参与的一个关系。实体 Bean 的域提供一个内部类表示该域的值。根据实体 Bean 的部署描述信息控制对该域的访问,例如在执行修改请求时,不能修改一个只读的域,不能修改主键等。同时根据 EJB 规范,实体 Bean 关系的修改必须遵循某些原则,这些原则也是通过 CMRFieldBridge 的控制逻辑实现的。

3)为了独立于底层数据库,当实体 Bean 的业务逻辑需要查找相关实体 Bean 时,必须调用使用 EJBQL 实现的查询方法。EJBQL 是专用于实体 Bean 查询的类似于 SQL 的编程语言。持久化管理器将 EJBQL 编译成数据库能够理解的 SQL 语句,根据运行环境设置好参数,然后提交给数据库执行。

4)InstanceContext 代表实体 Bean 在内存中的一个实例。在 JVM 中,不同的实例共享类型的方法,但保存私有的数据备份。对于实体 Bean 而言,私有数据备份保存在 InstanceContext 中。该类有两种活动实例:一种是无 ID 的状态,对 Home 接口定义的方法的访问使用这种实例;一种是有 ID 的状态,对 Object 接口定义的方法的访问需要使用这种实例。

5)PersistenceManager 除了解析部署描述信息,创建动态模型外,还对 EntityModel 进行封装,根据 EJB 规范对实体 Bean 的生命周期进行控制。

### 4.2 实体 Bean 动态模型的响应过程

实体 Bean 动态模型的各个部件的协作图如图6所示。

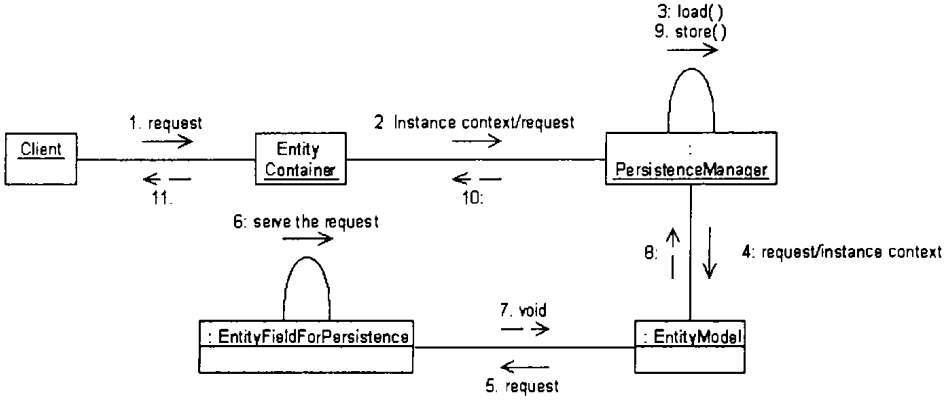


图6 实体 Bean 实例的使用

(1)客户端发出一个请求,它可以是对实例的读写操作;(2)表示实体 Bean 容器接受到这个请求,并为该请求生成一个实例 instance context;(3)持久化管理器收到发来的请求,首先将数据库数据装载到内存动态模型中;(4)持久化管理器向内存模型转发请求;(5)表示内存模型将请求转交给具体的某个或者几个域;(6)表示该域根据自身的控制逻辑执行请求;(7)处理结果返回到动态模型,动态模型进行必要的处理;(8)动态模型将处理结果返回到持久化管理器;(9)如果有修改的话,持久化管理器保存所作的修改;(10)持久化管理器将结果返回到容器;(11)容器对持久化管理器返回结果进行包装,以便返回给客户。

### 4.3 实体 Bean 数据的同步机制

如前所述,实体 Bean 的实例有两种类型,即有 ID 的实例和无 ID 的实例。无 ID 的实例对应关系数据库的一个表,而有 ID 的实例则对应关系表中的一个元组。在对有 ID 的实例进行访问时,实例的属性值对应关系元组的属性值。出于效率上的考虑,实际应用中并不需要这两组值任意时刻完全相等,而只是在一定的条件下才认为有必要进行同步更新的操作。

同步机制根据同步更新时机执行同步更新操作,并对同步更新操作进行优化。同步更新对应两个操作:load,从数据库中装载数据;store,向关系数据库写入内存数据。

首先是同步更新的时机。执行 load 操作的时机是当被访问的内存实例的某个域失效时。导致内存实例失效的事件有如下一些:使用实例的事务提交时,根据实体 Bean 部署描述信息的提交选项,EJB 规定了 *A*、*B*、*C* 三种选项;根据读入内存的时间,每个域都保存一个最近更新的时间,实体 Bean 的提供者指定一个有效期限,当这个时间超过这个期限,便认为该域已经失效;此外如果事务回滚,操作出现错误,该实例的所有非主键域都会失效[2]。执行 store 操作的时机是当内存实例的修改生效时。内存实例的修改生效的唯一途径是修改所在事务的提交。

同步机制还对同步更新操作进行仔细的优化。为了提高 load 操作的效率,同步机制使用惰性装载和提前装载的策略。惰性装载对域区别对待,有的域先装载,有的域在访问到时再装载。提前装载是指,在数据库装载时一次读入较多的数据元组,放入缓冲区中,需要进行新的装载时,先检查要装载的实例是否存在于缓冲区中,如果已经存在则不需要读取数据库。同样为了提高 store 操作的效率,同步机制检查实例中所有被修改了的域,将其写入到关系表中,而没有修改的域则不需要写入。

## 5 实体 Bean 容器的并发控制

前面讲述的持久化管理器管理考虑的仅限于单线程访问实例的情形。但实际上实体 Bean 通常运行在分布式环境中,对于客户端的每个访问线程都需要在服务器端生成一个线程为之服务,所以实体 Bean 的某个实例可能接受多线程的并发访问,此时可能出现如下错误:丢失更新,未提交依赖,不一致的结果[6]。为此实体 Bean 容器提供加锁方法,客户访问包含在一个事务之中。任何一个事务在获得对一个实例的访问权时,必需询问该实例相关联的锁。该锁根据自己的并发控制算法,对来访的事务进行处理,或者通过,或者等待。

OnceAS 的实体 Bean 容器提供两种锁:行锁和读写锁。行锁对数据库中每行数据加锁,当对应该行数据的实体 Bean 实例已经存在于内存中,所有对这行数据的访问都会被阻塞,只有当访问该行数据的事务提交,该锁才会被释放。行锁是通过数据库加锁来实现的,通过执行 select for update 语句,获得对一行数据的排它访问。读写锁的并发度较高,它将对数据的操作分为读写两种类型,对这两种类型的线程分别对待:多个读的线程可以共享一个锁,但是写锁必须独立访问。如果读线程没有释放,写线程必须阻塞直到所有的读线程都完成;如果有写线程在执行或者被阻塞,读线程必须阻塞直到写线程事务结束。唤醒阻塞线程时有两种方式,一种唤醒所有的线程,重新开始竞争,一种唤醒排在队首的进程。

在使用上述的锁时,一个事务占有某个实例,其余事务必须等待,直到该事务提交或者回滚。根据实体 Bean 的运行特点,一个事务可以申请多个锁,因此可能导致死锁。并发访问控制机制用一个二分有向图表示事务与锁之间的关系,事务占有锁则加入锁到事务的一条有向边,事务申请锁则加入事务到锁的一条有向边。如果某次申请导致环的产生,则自动回滚该事务。以图7为例说明这一过程。

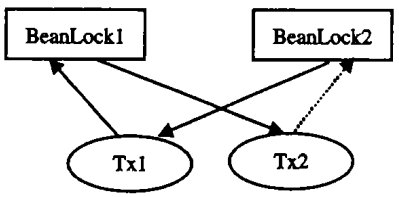


图7 导致死锁情形

在图7中有两个锁:BeanLock1和 BeanLock2,Tx1占有 BeanLock2,并申请 BeanLock1;Tx2占有 BeanLock1,准备申

请 BeanLock2。如果申请成功，那么必然存在一个 Tx1-BeanLock1-Tx2-BeanLock2-Tx1的环，出现死锁。此时唯一的方法是事务超时回滚，造成资源的浪费。但是并发控制机制能够察觉到死锁的出现，因此 Tx2对 BeanLock2的申请不会成功，而会导致它的回滚。

使用加锁机制，虽然牺牲了持久化管理器的效率，增加了复杂度，但是保证了实体 Bean 的可靠性，确保复杂的网络环境中实体 Bean 也能正确地工作。

**小结** 本文介绍了实体 Bean 容器对 CMP 的支持。为了支持 CMP，实体 Bean 容器需要提供持久化管理器。持久化管理器实现实体 Bean 的静态映射和动态映射。本文描述的持久化管理器使用较好的映射模型，并且具有较高的效率。此外，实体 Bean 容器提供并发控制机制管理对实体 Bean 的并发访问。本文描写的实体 Bean 容器和 CMP 持久化管理器已在中科院软件所自主开发的 J2EE 应用服务器[7]中实现。下一步的工作包括扩展 OR 映射模型，以体现更多的 OO 编程的思想；扩展对数据库的访问接口，以更好地支持数据库的新特性。

## 参 考 文 献


1 Sun Microsystems. Java 2 Platform Enterprise Edition Specification, v1.3. (2001/3/30)
2 Sun Microsystems. Enterprise JavaBeans Specification, Version 2.1. (2003/6/3)
3 Buschmann F, Meunier R, Rohnert H, et al. A Systems of Patterns: Pattern-Oriented Software Architecture. New York: John Wiley & Sons Ltd, 1996
4 Keller W. Mapping Objects to Tables-A pattern language, 2003, 7
5 Ambler S W. Building Object Applications That Work-Your Step-by-Step Handbook for Developing Robust Systems With Object Technology. New York: SIGS Books/Cambridge University Press, 1998
6 Ambler S W. Process Patterns: Building Large-Scale Systems Using Object Technology. New York: SIGS Books/Cambridge University Press, 1998
7 范国闯. Web 应用服务器关键技术研究


(上接第170页)

大文件(10M)的时候，Dynamic 的请求次数是 adPD 的5.1倍。大量的数据请求不仅给服务器带来了较大的负载，也使得下载性能大大下降。

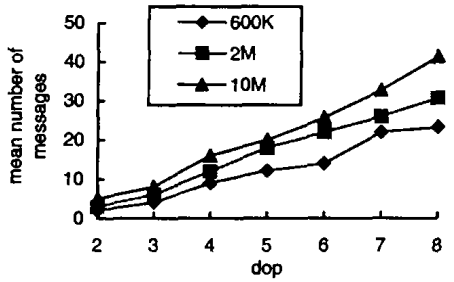


图2 不同文件大小情况下 adPD 平均数据请求数量随 dop 值变化图

在3.1节中我们提到了采用 pipeline 的方法避免数据请求带来的连接空闲。为了验证 pipeline 带来的性能提高，我们对 Dynamic 和 adPD 在文件大小为10M 的情况下使用了 pipeline 方法，试验结果见图3。从图中可以看到，使用 Pipeline 之后，Dynamic 性能提高明显，平均等待时间下降了不少，但是性能仍然没有超过没有使用 pipeline 的 adPD 算法。这主要是因为网络节点间的 RTT 值变动比较大，这给精确估计发送请求的提前量带来了困难，使得一部分请求的延迟没有完全被 pipeline 消化。而使用 pipeline 并不能给 adPD 带来多少性能提高，这主要是因为 adPD 的数据请求数量已经相对较少，pipeline 能够节省下来的空闲时间不是很多。考虑到使用 pipeline 给算法增加了额外的复杂度，我们认为在实际使用中一般的 adPD 算法就已经足够了。

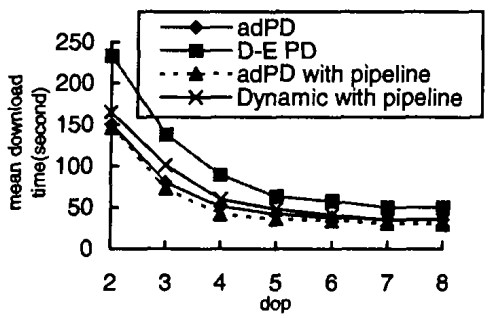


图3 使用 pipeline 前后 adPD 与 Dynamic 性能对比图 (F=10M)

**结论** 本文提出了一种新的速度自适应的动态并行下载机制--adPD。adPD 通过为速度不同的连接动态分配大小不同的下载任务，可以很好地适应传输连接速度的变化，做到按速度比例分配下载任务量，充分利用带宽。同时，通过划分大小不固定的文件分块，adPD 还可以尽可能地减少发送数据请求的数量，在减轻提供服务的节点的负载的同时，提高了下载速度。实验证明 adPD 与已有的几种并行下载算法相比，性能提高明显，缩短了下载时间，同时也大大减少了发送的数据请求的数量，是一种高性能的并行下载算法。

## 参 考 文 献


1 Mydrivers' web site. http://www.mydrivers.com
2 Akamai's web site. http://www.akamai.com
3 Gnutella. http://gnutella.wego.com
4 Rodriguez P, Kirpal A, Biersack E W. Parallel-access for mirror sites in the internet. In: Proc. of IEEE INFOCOM 2000, March 2000
5 Miu A, Shih E. Performance Analysis of a Dynamic Parallel Downloading Scheme from Mirror Sites Throughout the Internet. url: http://nms.lcs.mit.edu/?aklmiu/comet/paraload.html, December 1999
6 Speedbit's download accelerator. http://www.speedbit.com
7 Cohen B. Incentives Build Robustness in BitTorrent. http://bitconjurer.org/BitTorrent/documentation.html, May 2003
8 OpenCola Swarmcast. http://www.opencola.org/projects/swarmcast.shtml
9 Myers A, Dinda P A, Zhang H. Performance characteristics of mirror servers on the internet. In INFOCOM (1), 1999. 304～312
10 Gkantsidis C, Ammar M, Zegura E. On the Effect of Large-Scale Deployment of Parallel Downloading. In: IEEE Workshop on Internet Applications (WIAPP'03), 2003
11 Rodriguez P, Biersack E. Dynamic parallelaccess to replicated content in the Internet. IEEE/ACM Trans. on networking, 2002, 10(4)
12 Koo S G M, Rosenberg C, Xu D. Analysis of Parallel Downloading for Large File Distribution. In: Proc. of FTDCS 2003, San Juan, Puerto Rico, May 2003
13 Zeitoun A, Jamjoom H, El-Gendy M. Scalable Parallel-Access For Mirrored Servers. In: The 20th IASTED Intl. Conf. on Applied Informatics (AI 2002), Innsbruck, Austria, Feb. 2002
14 Sayal M, Breibart Y, Scheuermann P, Vingralek R. Selection Algorithm for Replicated Web Servers. In: Workshop on Internet Server Performance, SIGMETRICS, Madison, USA, June 1998