计算机科学 2005Vol. 32№. 12

基于组件架构的对象持久层框架设计

曾 一 徐 珂 李 颖
(重庆大学计算机学院 重庆 400044)

摘 要 本文给出了一种基于组件的对象持久化设计框架。该框架不同于以往在持久层设计中简单将整个 DMBS 的功能复制到持久层中,而是将 ORM(Object-Relational Mapping:对象-关系映射)独立的功能用单独的组件实现,并将这些组件分为四个层次。框架中的组件之间是相互独立的,可以单独使用。构建持久层时,这些组件能够良好地结合在一起协同工作,并能够根据不同的需求定制持久层。本文提出的设计框架为对象持久层的实现提供了一种基于组件的、可定制的并易于管理和实现的途径。本框架可作为对象持久化管理的一个设计模式。
关键词 对象持久层,中间件,组件,ORM

A Component-Based Design of Object Persistence Layer

ZENG Yi XU Ke LI Ying
(College of Computer Science, Chongqing University, Chongqing 400044)

Abstract A component-based model of object persistence layer is introduced. Instead of simply copy DBMS functions to the database middleware, the paper separates the core of the ORM functions into several independent components and puts them in 3 layers. Each of the components is self-contained and can be used separately. While configured together, they are fine-grained and can be customized easily for particular requirements.
Keywords Object persistence layer, Middleware, Component, ORM

1 问题的提出

近年来发展迅速的面向对象技术,为应用系统开发提供了强有力的理论和技术支持,而与应用系统开发息息相关的数据库技术,却仍然是关系数据库理论一枝独秀。面向对象数据库管理系统(OODBMS)由于各种各样的原因[1],一直无法成为数据存储市场的主流。面向对象技术是现实世界的抽象,关系数据库是基于严格的关系理论的二元表结构存储系统,这就造成了实际应用中二者协同工作时出现诸多的问题。

目前,在持久层方面的研究大多将持久层看成应用程序的一个组件,该组件提供了 DBMS 部分甚至全部功能,这使得持久层本身结构复杂、耦合性高且不易实现,或者实现后的持久层功能过于集中、复杂,增加了应用难度。本文提出的基于组件架构的持久层,则可视为一组集成良好的组件,每个组件提供单一的功能,如日志记录、缓存、并发控制等。这些组件既可以为满足特定需要而单独使用,又可以良好地结合在一起协同工作,从而达到了降低耦合、简化实现、易于配置的效果。

2 设计框架概述

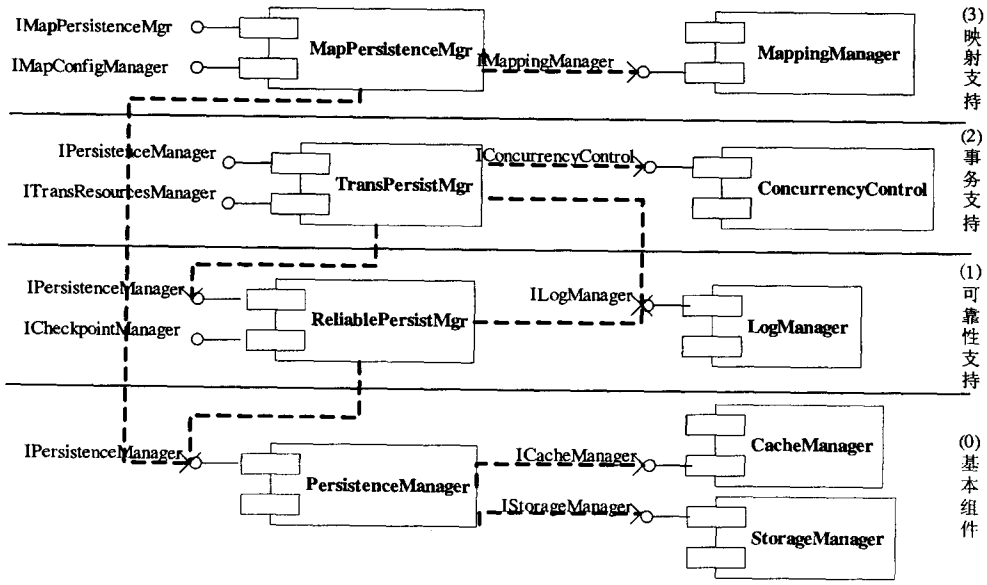


图 1 框架结构图

针对上面提出的问题,参照 OODBMS 的实现机制,运用面向对象的方法,使用 UML,提出了一个基于组件的、可定制的四层结构的持久层框架,如图 1 所示。该框架分为四个层次,除了映射支持层,其他的每一层扩展下一层提供的功能。基于这种方式,根据相应的需求,可以用某一层或多层中的相关组件来定制持久层。该框架的设计遵循以下三个原则:(1)

组件可以为满足特定需要而单独使用(Self-contained);(2)其中的组件可以作为持久层的组成部分,并可以定制(Customizable);(3)基于组件的结构是为了降低持久化管理中的依赖关系,减少耦合(Fine-grained)。

如图1所示,本框架中,基本组件层是整个持久层的核心,为持久层提供基本的缓存管理、存储管理和同步控制;可靠性支持层扩展基本组件层,增加日志管理,为持久层提供系统失败时的可靠性支持;事务支持层扩展可靠性支持层,在日志的基础上增加重做(Redo)管理和设置检查点,提供并发控制和事务支持。这三层构成了一个功能完善的持久化管理层。映射支持层是在前三层的基础上增加编程开发时的映射生成、映射管理功能。

框架中的缓存管理器、存储管理器、日志管理器、并发控制管理器、映射管理器是可以独立使用的,其余的组件可以被视为框架的"粘合剂",为整个持久层提供一个协作协议。映射支持层可以看做是持久层的上层应用,在实际的实现中可与其它层的组件搭配,用以提供ORM[1]功能。由此可见,本框架可作为对象持久化管理的一个设计模式。

3 基本组件层(0层)设计

0层是整个框架的核心,这一层的组件组成了一个无可靠性支持的、最基本的持久化管理层。本层的基本组件有:

· 缓存管理器(CacheManager)。负责将常用的数据对象保存在缓存中。

· 存储管理器(StorageManager)。负责为底层数据库存储提供一个通用接口,提供对多种底层数据库的支持。

· 持久化管理器(PersistenceManager)。负责控制和协调缓存管理器和存储管理器、管理对象的持久化。该组件不包含任何系统恢复相关的功能。

本层负责从数据存储中读取或写入数据对象,图2给出本层的组件接口图。

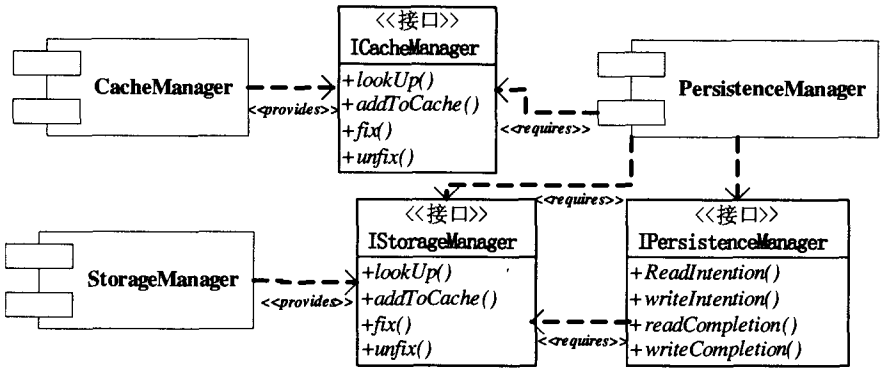
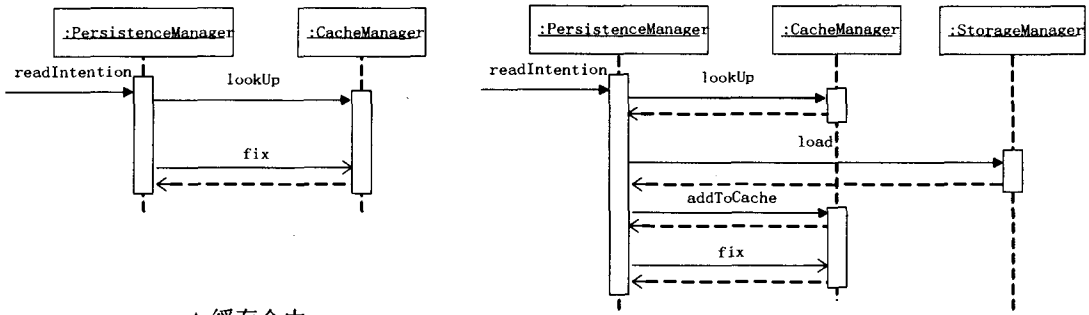


图2 0层组件图

本层应用的前提是系统失败(包括应用程序失败和数据库系统失败)不会对持久化数据产生负面影响,因此可以用于仅仅对数据库进行查询操作。

3.1 缓存管理设计模式

持久层不仅仅是在应用程序对象和数据库建立简单的联系,为减轻数据库负载,提高应用程序效率,基本的缓存管理必不可少。因此,本框架将缓存管理作为持久层的基本功能,放在基本组件层。持久层为使用最为频繁的数据对象建立拷贝,由缓存管理器(CacheManager)管理。



A.缓存命中　　B.缓存命中失误

图3 缓存管理时序图

图3给出了该框架的缓存管理控制时序图,整个序列开始于缓存管理器中的lookUp(查找)方法,如果查找到了请求的对象,缓存管理器仅仅使用fix(修复)方法将这个数据对象修复,然后返回给持久化管理器(图3中A图)。如果缓存管理器没有找到被请求的数据对象,持久化管理器就调用存储管理器的load(加载)方法,请求该数据对象的一个拷贝。得到该数据对象后,持久化管理器会依次调用缓存管理器的addToCache(加入缓存)和fix方法,将该数据对象载入缓存并修复(图3中B图)。

3.2 同步控制

数据对象与底层数据之间的一致问题,也就是同步问题,是持久层核心必须提供的另外一个基本功能。本框架设置了一个脏对象管理器(DirtyObjectManager)进行同步控制,并利用一组监听器监听需要同步的操作。

同步控制可以分为两类:立即更新和异步更新。立即更新简单而易于实现,但会带来很大的系统开销。为实现异步

更新，持久层必须提供一个底层的恢复机制。另外一个重要的问题是要判断引发同步的源头，这涉及到组件之间的依赖问题。针对以上两个方面，提出本框架的同步策略：

①该框架中用“脏对象表”来管理脏对象（未与底层数据库同步的数据对象）。脏对象表由脏对象管理器（DirtyObjectManager）维护和管理。

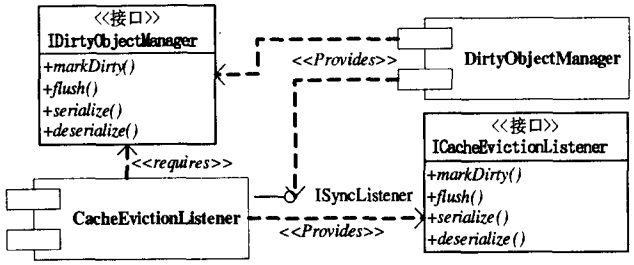


图 4　同步控制组件图

②脏对象管理器（DirtyObjectManager）依赖一组同步控制触发器（ISyncListener）来监听和处理同步。

③当一个同步控制触发器被触发，该触发器就会请求脏对象管理器检查目标数据对象。如果该数据对象为脏对象，就立即更新该数据对象，否则把该数据对象丢弃。

图 4 给出了引入同步控制后的持久层框架组件接口图。

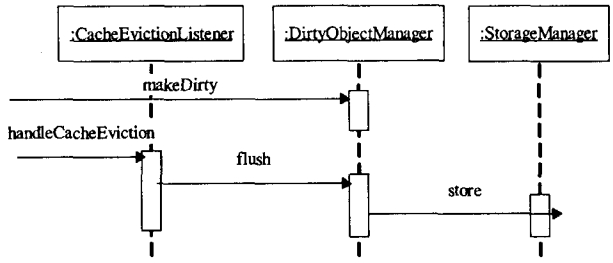


图 5 同步控制时序图

在本框架的 0 层即无可靠性支持的持久层中，缓存事件监听器是唯一的同步控制触发器。图 5 给出了这种情况下同步控制的时序图。除了缓存回收事件，还应有以下系统事件触发同步控制：

• 检查点设置。系统使用检查点设置进行系统失败后的恢复，检查点设置的目的是减少数据恢复和清空日志的时间消耗。需要注意该事件应该更新相关的脏数据对象。

• 事务提交。应该在事务提交时强制进行同步控制，所有被该事务更新的数据对象都应该同时同步到底层数据库中。

• 重新连接。在无线应用等一些弱连接环境中，连接中断后又重新连接时，应该首先进行同步控制。

• 应用程序事件。持久层应该为应用程序提供进行同步控制的接口，允许应用程序在必要时直接介入更新控制。

4　可靠性支持层（1 层）设计

本层（即 1 层）为整个持久层框架提供基本的可靠性支持。系统失败时的可靠性支持是解决在应用程序的数据库操作或数据库系统发生失败后，将数据对象持久化到底层数据库时的问题。为解决这类问题，本层设有 2 个基本组件：

• 日志管理器（LogManager）。负责记录和维护日志，提供一组日志管理接口。

• 可靠性持久化管理器（ReliablePesistenceMgr）。扩展了基本组件中的持久化管理器，依靠日志管理器（LogManager）采用重做（redo）方法为持久层提供恢复支持。

本层使用可靠性持久化管理器为系统失败时提供可靠性支持，该组件利用日志管理器保存一份数据库操作的日志，并利用日志进行重做（redo）来实现。增加了可靠性支持后的框架组件图如图 6 所示。

4.1　收集重做（Redo）信息

为支持异步更新，持久层需要保存重做恢复信息（Redo Recovery Information）。在本框架中，这些信息由可靠性持久化管理器在 writeCompletion()（写操作完成）方法被调用时收集，如图 7 所示。完成日志记录后，writeCompletion()方法被传递到 0 层。

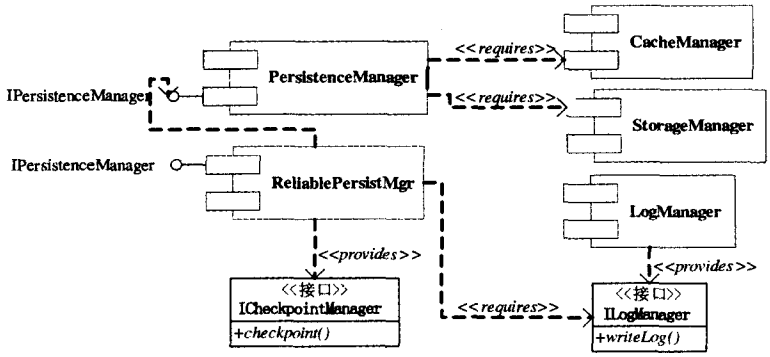


图 6　增加了可靠性支持后的框架组件图

4.2　检查点设置

设置检查点可以减少在系统失败后恢复工作时的工作量。检查点设置的前提是其之前的所有更新操作都是稳定可靠的。必须注意，检查点设置会导致同步操作。一般而言，检查点设置可以分为以下 3 级[2]：

• 强制检查点设置。停止一切其他活动，将缓存中所有已更新对象同步到底层数据库，记录一个“检查点设置完成”日志。

• 递增检查点设置。这种方式下对脏对象表的管理是分批进行的，每一批与相应的检查点间隔对应。在这种情况下，只有最早的检查点间隔中的脏对象被同步到数据库。

• 轻量级检查点设置。使用遍历撤销/重做恢复信息的方式来实现，仅记录系统数据结构，如脏对象表和活动事务表等。

图 8 为本框架中典型的检查点设置时序图。其中，强制检查点设置需要执行其中的第 2 步和第 4 步；递增检查点设

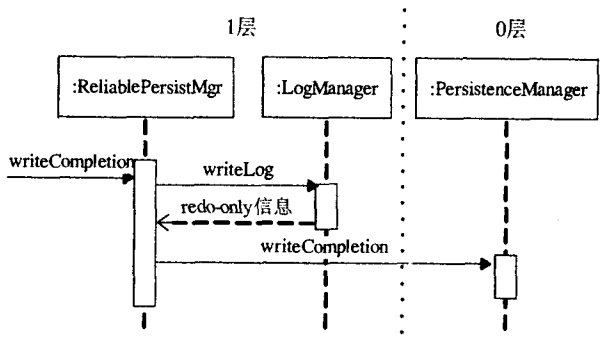


图 7　重做信息收集时序图

置需要执行其中的第 1、2、4 步；轻量级检查点设置执行其中的第 1、3、4 步。

5　事务支持层(2 层)设计

2 层为持久层框架提供事务支持。事务有三个基本属性，即独立性、原子性和持久性。原子性和持久性通常使用单一的恢复(redo)方法来实现，但为保证组件之间的独立性，本框架用两个相互独立的组件分别实现。到此为止，已经给出了一个基于多级恢复(Mutil-level Recovery)[3]的三层持久层核心框架。以下为本层的基本组件：

·并发控制管理器(ConcurrencyControl)。提供一组并发控制方法的接口。

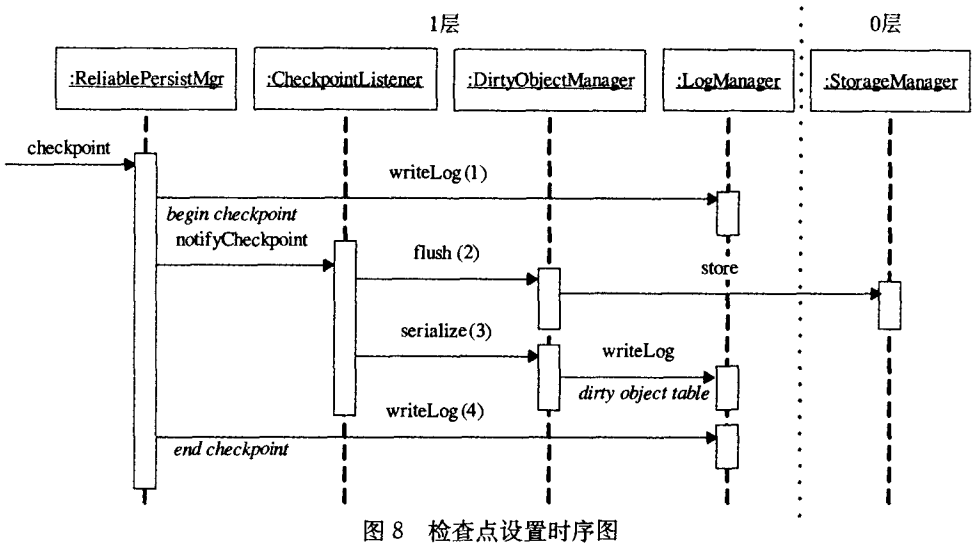


图 8　检查点设置时序图

·事务持久化管理器(TransPersistMgr)。为整个持久层框架提供事务支持。该组件依赖可靠性持久化管理器(PersistenceManager)和日志管理器(LogManager)实现事务的原子性和持久性，依赖并发控制管理器(ConcurrencyControl)实现事务的独立性。

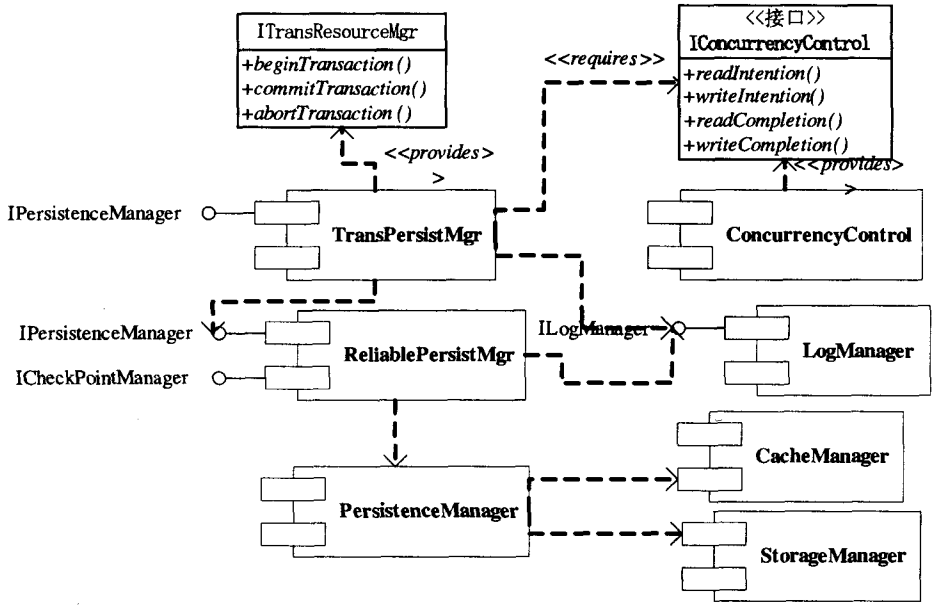


图 9　增加事务支持后的组件图

图 9 给出了增加了事务支持后的持久层框架的组件图。其中的事务持久化管理器(TransPersistMgr)起到调度器的作用。事务的独立性由并发控制管理器(ConcurrencyControl)体现；持久性由可靠性持久管理器(ReliablePersistMgr)体现；原子性由事务持久化管理器(TransPersistMgr)使用多级恢复方法实现。

图 10 给出了事务操作的时序图。首先，事务开始于 beginTransaction( ) 的调用，事务持久化管理器(TransPersistMgr)首先请求日志管理器(LogManager)在日志中记录一条“事务开始”记录。对于每一个被事务访问的数据对象，在其

被更新之前都要调用一次 writeIntention()方法，以便事务持久化管理器调用日志管理器记录一条撤销日志。重做信息由1层的可靠性持久化管理器(ReliablePersistMgr)调用日志实现，前面已经提到。需要注意的是，当 writeCompletion()方法被调用的时候，1层的操作就已经完成。事务处理时序结束于 commitTransaction()调用，日志管理器记录一条记录，以备发生失败后的重做。随后，writeCompletion()方法被传递到0层的持久化管理器(PersistenceManager)，最后事务持久化管理器请求日志管理器记录一条"事务完成"的日志记录。

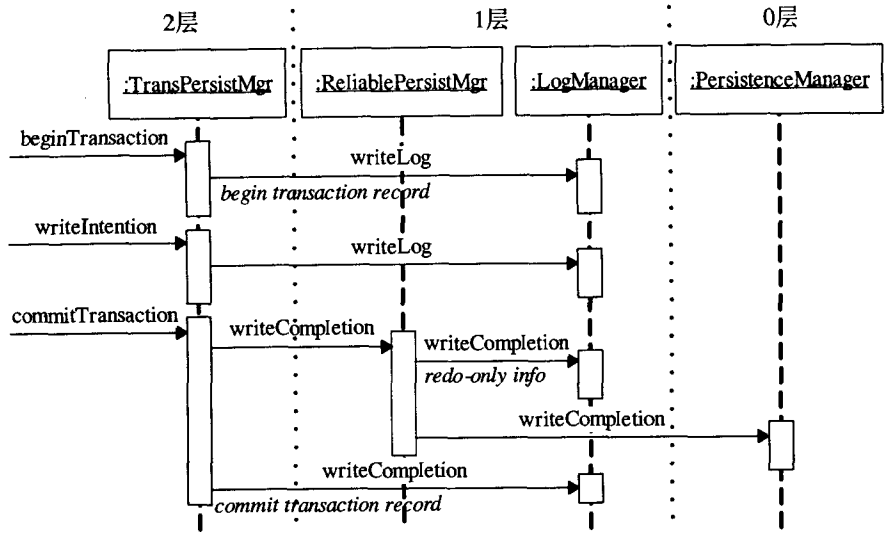


图 10　事务操作时序图

多级恢复方法：数据库系统失败后，持久层与数据库的连接重新建立时，1层记录的重做(redo)信息首先被无条件执行，即搜索并执行持久层框架中1层记录的未完成的重做日志。随后检查2层中记录的未完成的撤销(undo)日志，并执行相应的撤销操作。这就是本框架中的多级恢复(Muti-level Recovery)[3]方法。

6　映射支持层(3层)设计

映射支持层是对基本组件的应用扩展。以下为本层的基本组件：

·映射管理器(MappingManager)。为整个框架提供映射策略支持，提供一组应用层次对象-关系映射关系管理的接口。

·映射持久化管理器(MapPersistenceManger)。为整个持久层提供应用层面的对象-关系映射支持，该组件扩展了基层的持久化管理器(PersistenceManager)，提供应用映射配置管理器接口(IMapConfigManager)，并依赖映射管理器(MasppingManager)实现映射管理。

在基本组件的基础上，通过映射管理器(MappingManager)提供的映射标准管理器(IMapCriteriaController)接口，定义在数据库和应用系统数据对象之间映射的标准。有关映射策略、映射管理等参见文[1,4]。PersistenceObject 对象是所有数据对象的一个基类，依赖 IMapCriteriaController 接口，被映射持久化管理器(MapPersistenceMgr)管理。

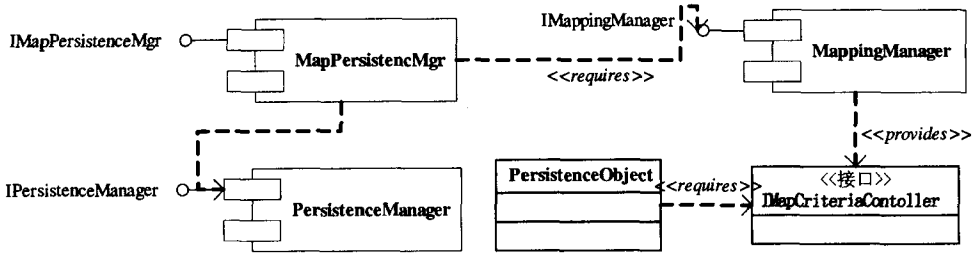


图 11　映射支持组件图

结论　整个框架是一个基于组件的、可定制的、功能相对完善的对象持久层框架。该设计框架很好地解决了持久层设计中功能过于集中、不易实现、不易应用的困难。在搭建实际应用的持久层时，可以根据实际需要从本框架中选择0层、1(0+1)层、2(0+1+2)层定制实际需要的对象持久层。结构中的3层提供对应用开发时的映射支持，在实际应用中，可以根据需要选择使用。

本框架在开发某大型管理信息系统的应用过程中，取得了较高的灵活性和性能，收到良好的效果。

参 考 文 献

1　Henning M. A New Approach to Object Oriented Middleware. IEEE Internet Computing, 2004. 3～4
2　Garcia-Banuelos L, Duong P, Nedelec T, et al. Experienceing Persistence Object Management Customization (Demonstration). In: Proceedings of the DBA'02 French Conference, Oct. 2002
3　Lomet D B. MLR: A Recovery Method for Multi-level System. In: Proceedings of the SIGMOD Conference, 1992
4　Liberator JDO 开发手册(Dev Guide for JDO 1.2). http://www.redsoftfactory.com/docs/ref/index.html. 2004.10
5　Larman C 著. UML 和模式应用(第二版). 姚淑珍，李虎，等译. 北京：机械工业出版社，2002