对象持久层的设计*

衡冬梅[1]，唐拥政[2]，朱跃龙[1]
(1. 河海大学 计算机及信息工程学院，江苏 南京　210098；2. 盐城工学院 现代教育技术中心，江苏 盐城　224003)

摘　要：描述了面向对象应用与关系数据库之间对象持久层的设计，提出了针对基本维护性数据的改进持久层性能的方案，并分析了这种方案的可行性。

关键词：持久层，性能，缓存镜像表
中图分类号：TP311　　**文献标识码**：A　　**文章编号**：1671－5322(2005)03－0045－04

面向对象的数据库应用开发中一般有两种策略，即基于面向对象的数据库的开发和基于关系数据库的开发[1]。由于面向对象的数据库技术还不成熟，实际开发中大多还是采用基于关系数据库的策略。这样，就必须用持久层[2~3]实现面向对象数据库的操作，以处理关系数据库中各个表之间的错综复杂的关联关系。

对象持久层的功能实现并不复杂，关键在于建立对象模型和关系模型之间的映射和查询转换，从设计模式和持久化框架不同的侧面来解决这个问题[4]。对象持久层在实际应用中的难点在于如何保证系统的性能，因为持久层在提高应用系统的可维护性、可扩展性和可移植性的同时，也存在这样的问题：对应用系统的性能有所影响，存取数据时速度与效率会比原先直接使用 SQL 的系统稍慢，因此本文主要讨论如何对持久层进行合理设计来提高性能。

1　对象持久层的框架设计

持久化层实际上就是封装了对象持久化功能的一组类。目前较为流行的持久层框架有 TopLink，Castor，OJB，Hibernate 等，本文中设计的持久化层主要参考轻量级持久化框架 Hibernate，实现对象持久化的基本功能，如类到数据库表的映射，还提供数据查询和获取数据的方法，该持久层的设计包含了 6 个大类。在表 1 中对这 6 个大类功能做了具体描述。

使用该持久层时，应用程序的程序员只需要知道 PersistentTransaction，PersistentBroker 提供的接口就可以了。

表 1　持久层类的描述
Table 1　Description of the class of Persistent Layer

PersistentObject	PersistentObject 封装了使单个对象持久化所必须的行为，所有需要持久化的对象都要根据 PersistentObject 的 Operator 的方法规则重载操作符。
PersistentTransaction	PersistentTransaction 封装的对多个对象持久化的行为，包括多个对象的存储、更新、删除；并且封装了数据库事务操作的所需的行为。
PersistentBroker	PersistentBroker 是需要持久化的对象和持久化机制之间的连接，处理对象和持久化机制之间的通信，提供存储，更新，删除，恢复等操作给用户，它是用户和持久化机制之间的桥梁。
ClassMap	ClassMap 是映射类，它的功能是将对象的属性映射成数据库中的库表。
Sql 类	Sql 类是封装了写库的 sql 语言，把 sql 语句封装起来，可以更好的体现程序面向对象的特点，且模块化程度更高
PersistentMechanim	PersistentMechanim 封装了对持久化机制的操作方法

* 收稿日期：2005－07－11
作者简介：衡冬梅(1981－)，女，江苏建湖县人，河海大学计算机及信息工程学院硕士研究生。

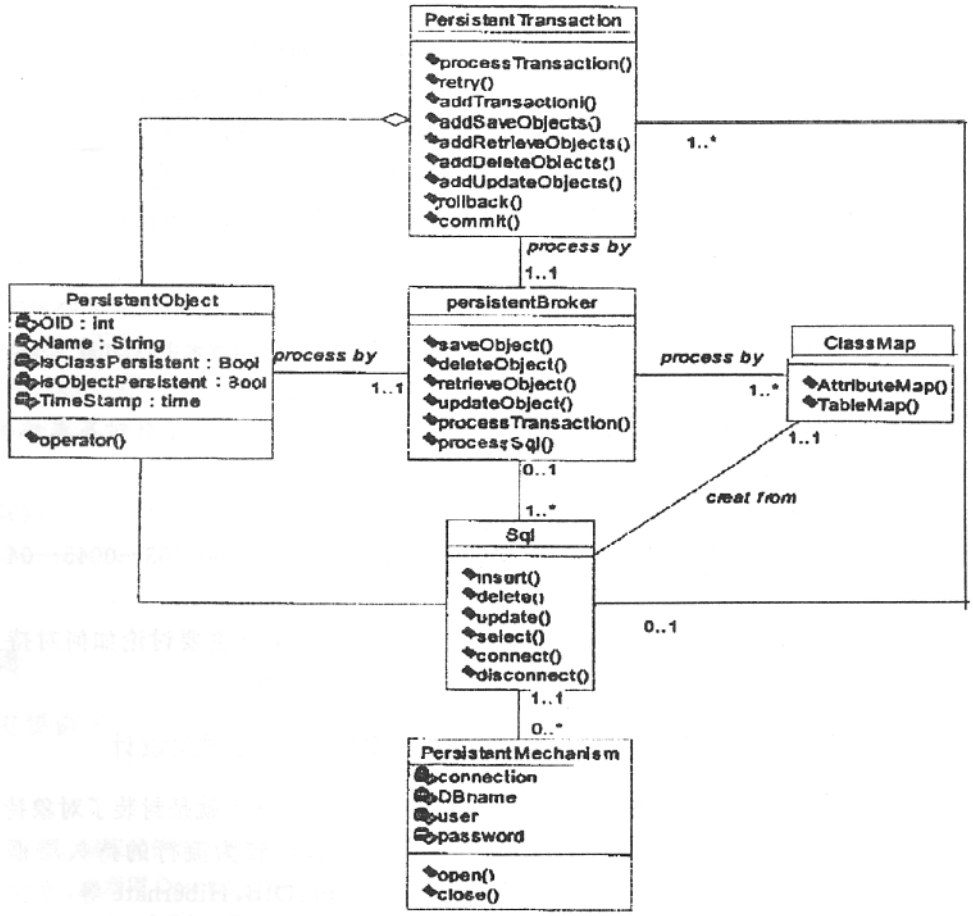

图 1 持久化层的框架设计

Fig. 1 The design of the framework of Persistent Layer

2 持久层性能优化

2.1 缓存镜像表的提出

在一个公司信息管理系统中,员工表-部门表是关联表,员工表是 DepartmentID 关联部门表的主键。如图 2 所示。要求:显示员工名时,显示

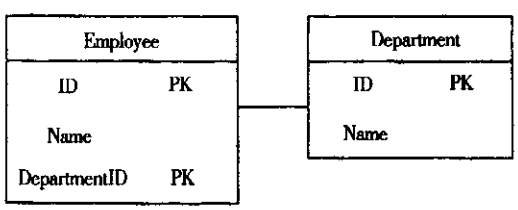

表 2 员工表-部门表

Table 2 Employee—department

此员工所属部门名称。在传统非 ORM 模式下,可采用 Select 语句来读取,这种非 ORM 模式开发的系统,系统扩展性、数据库移植性都比较差、开发效率低下。

在 ORM 模式下,这两个表映射为 EmployeeEntity 员工类与 DepartmentEntity 部门类。当 Retrieve 到员工类的一个员工对象 employee1 后,如要求显示部门名,就需要 DepartmentEntity 去 Retrieve 对应的部门 department1,通过使用 department1 的 Name 属性显示部门名。这个过程中进行了两次数据库访问,假设此员工表有 5 个关联的外键,那么一个员工的显示,将进行 6 次数据访问,则数据库访问的性能大打折扣。

对于这种情况,现有的持久层中的解决方案是:在 XML 中配置 5 个 one—to—many 的映射关系,然后采用非 Lazy loading(延迟加载)的方式进行一次性读取,这样就把 6 次的数据访问一次性执行了.但如果有 N 个人同时进行这样的操作,那么数据访问数还是非常大,达到 $N\times 6$ 次。

对于上面的情况,我们可采用另一种更好的方式解决。我们可以把这个部门表的内容放到缓存里,形成一个"缓存镜像表",即数据库中实际表的一个镜像,实时保持与实际表的同步,判断如下:

if(缓存中镜像表存在)

从缓存直接读取;

else

{从数据库读取;

建立缓存镜像表,把数据库中的数据放到缓存中;}

这样的方式可以减少很多的数据访问,就拿上面的例子来说,在第一个人进行了 6 次访问后,以后的 N 个人执行,只要执行 N 次数据访问了,速度将会有一个突破性的提高。

2.2　缓存镜像表的可行性分析

对于这种"缓存镜像表"的可行性我们先从数据库数据来分析。一个系统将包括很多的数据,我们可以对这些数据分一下类:

(1)维护型数据:这是指那些在系统中的基本数据,比如前面的部门类型、客户类型、付款方式、结算方式、销售类型、货币种类等等,这些数据存在的共同点是:数据量小、字段少、使用频率高、维护少。因此,具有这些特点的数据,我们就非常有理由放到内存中缓存起来。

(2)操作型数据:这种数据的特点是数据量比较大,字段也比较多,更改频繁,比如员工、客户等信息,不适合用到内存中。

(3)日志型数据:这种数据数据量更多,比如订单、订单明细、操作记录等,更不合适。

因此该方案主要是考虑"维护型数据"。

2.3　缓存镜像表的可操作性分析

在可行性分析后,我们要考虑,在实际操作起来要注意哪些。关键在于"如何保持实时的镜像关系",也就是要实时保持缓存中镜像与实际的数据库表数据一致。那么我们就要保证在进行更新(增,删,改)时能同时更新内存表数据就可以了。

在 OR Mapping 时,我们标识一个实体为"需保存为镜像",那么持久层(PersistenceLayer)在进行此对象的 Save()、Update()、Delete()时,自动进行"同步处理",而且这是完全可以实现的。

2.4　缓存镜像表的代码实现

在实体类的定义中增加代码:

＜member name＝"C:PersistenceLayer.ClassMap._IsSaveInCache"＞

＜summary＞用于判断此实体是否要保存到缓存中＜/summary＞

＜/member＞

＜member name＝"C:PersistenceLayer.DeleteCriteria._IsSaveInCache"＞

＜summary＞

用于判断删除操作是否要保存在缓存

false:不保存在缓存中。则每次都从数据库读取,为默认值

true:保存到缓存中,如果缓存中存在就直接从缓存中读取,对于基本数据建议使用此功能。对于频繁操作数据不推荐,这会导致大量内存被占

＜/summary＞

＜/member＞

同样的

＜member name＝"C:PersistenceLayer.InsertCriteria._IsSaveInCache"＞用于判断插入操作是否要保存在缓存

＜member name＝"C:PersistenceLayer.UpdateCriteria.IsSaveInCache"＞用于判断更新操作是否要保存在缓存

在这个持久层中有个静态类,它的 ArrayList 里存放各种"缓存镜像表"的"镜像 DataTable",在进行 Retrieve()时,先判断此 ArrarList 中是否存在此实体的"镜像",如果有的话,则直接从此"镜像 DataTable"中 Retrieve()出来,还有 RetrieveCriteira(获取标准)时,会从此"镜像 DataTable"中使用 Select 符合条件的,生成新的"镜像 DataTable"返回出来。实体在进行 Save()、Delete()和 Update 操作时都会进行"镜像 DataTable"的同步更新。

在应用中,要让实体保存到缓存中,只要在实体的 XML 配置文件中指明 IsSaveToCache＝true 即可。

如:

＜class name＝"DepartmentEntity" table＝"Department" database＝"Oracle9i" IsSaveToCache＝"true"＞

＜attribute name＝"Id" column＝"Id" type＝"Integer" increment＝"true" key＝"primary" /＞

＜attribute name＝"No" column＝"No" type＝"String" key＝"primary" /＞

＜attribute name＝"Name" column＝"Name" type＝"String" /＞

＜/class＞

3 总结

从上我们可以看出,效率的提高建立在内存的牺牲之上,如果有过多"维护型数据"使用"缓存镜像",整个内存消耗将非常大,所以,在考虑使用多少"缓存镜像表"时,要考虑服务器的承受能力的。但是,增加内存,提高性能这是开发过程中很多客户所追求的。我们只要能仔细分析,是可以得出一种好的"缓存镜像表"的。

参考文献:

[1] 车敦仁,周立柱. 关系数据库与面向对象数据库的集成[J]. 软件学报,1996,7(11): 669-675.
[2] Scott W Ambler. Process Patterns—Building Large—Scale Systems Using Object Technology[M]. Cambridge:Cambridge University Press,1998.
[3] Scott W. Ambler. The Object Primer 2nd Edition[M]. Cambridge:Cambridge University Press, 2001.
[4] 朱庆伟,吴宇红 . 一种对象/关系映射框架的分析和应用[J]. 电子科技, 2004(1): 54-57.

Design of Object Persistent Layer

HENG Dong-mei[1], TANG Yong-zheng[2], ZHU Yue-long[1]
(1. College of Computer and Information Engineering of HoHai University, Jiangsu Nanjing 210098, China
2. Modern Educational Technology Center, Yancheng Institute of Technology, Jiangsu Yancheng, 224003, China)

Abstract: This paper describes the design of an object persistent layer , which is a middle ware between object—oriented application and RDBMS. The authon proposes one solution to the improvement of the object persistence layer, which focuses on basic attendance data, and analyses the feasibility of this solution.

Keywords: persistent layer, capability, cache mirror table

(上接第 25 页)

参考文献:

[1] YANG Qun, ZHAO Yanan, A fuzzy logic—based framework for voute choice in vehicole navigation systems [J]. Journal of Systens Science and Systems Engineering, 2000, 9(4): 467—474.
[2] Shier RD. ON algorithms for finding the K Shortest paths in a metwork [J]. Networks, 1979, 9(9): 195—214.
[3] 陈芒,陈洪亮 . 智能交通系统中的路径牵引算法[J]. 微型电脑应用, 1999, 15(6): 45—46.
[4] 杨兆开,初连禹 . 动态路径诱导系统的研究进展[J]. 公路交通科技, 2000, 17(1): 34—38.
[5] 郑长江 . 动态交通分配方法及其初步应用研究[D]. 南京:东南大学, 2000.

Study on Replaceable Route Choice Approach

CHENG Yang, WANG Bo
(School of Management, Shanghai Science and Technology University, Shanghai 210000, China)

Abstract: The study on Pynamic Route Gvidance System(DRGS) is an important aspect in the field of Intelligent Transportation System(ITS). The core of the study is to find a "best route"for drivers according to the current traffic information. As a result the travel time can be saved and the traffic congestion can be avoided. This method can give drivers a single "optimal" path to follow. After the adoption of the improved K-shortest path algorithm, we can offer a number of routes with different features to drivers, which will tally with the as tual situation better.

Keywords: 万方数据—shortest path algorithm; replaceable route