基于 DAO 模式的持久模型的研究与设计①

孙 霞 杨润萍 骆 挺 霍 瑞 (宁波大学 科技学院 浙江 宁波 315020)

摘 要： 开发一个同时使用面向对象技术和关系型数据库技术的项目是一件困难的事情。提出了一个基于 DAO 模式的数据持久模型，通过可变机制将底层的数据访问独立封装起来，使得对象模型与关系数据库之间具有无关性。同时通过缓冲等机制大大提高了数据访问的效率。

关键词： 数据持久; DAO 模式; 面向对象; 关系数据库

Research and Design of a Data Persistence Model Based on DAO Pattern

SUN Xia, YANG Run-Ping, LUO Ting, HUO Rui

(College of Science and Techology, Ningbo University, Ningbo 315020, China)

Abstract: Designing a software to connect an object-oriented business system with a relational database is a tedious task. This article proposes a data persistence model based on DAO Pattern. It completely makes the access to underlying data independently encapsulated, and makes object model and relational database independent of each other by Variable Mechanism. In addition, the data accessing efficiency is greatly improved through the buffering mechanism.

Keywords: data persistence; DAO pattern; object-oriented; relational database

1 前言

当前主流的面向对象开发技术和关系数据库之间存在着阻抗不匹配(impedance mismatch),对象不能自然的与关系数据库中的数据交互。使用关系数据库的面向对象系统开发人员通常要花费大量的时间来将对象持久化。如果应用系统的对象持久化工作是由开发人员而不是设计人员完成，那么开发人员必须考虑诸如连接管理，数据访问效率、多线程并发、事务处理、不同系统之间的互操作问题，而且在应用系统的生命周期中，持久存储的类型有可能会改变，业务模型也有可能发生变更。设计人员提出了持久层概念，目的是隐藏持久化细节，而让开发人员专注于问题域而不是对象持久化。

2 DAO模式的工作原理

DAO(Data Access Object)模式是将业务逻辑从数据存取逻辑中分离出来，把存取的资源改编，从而使资源可以容易和独立地转变。如图 1 所示，业务逻辑并不直接和数据源交互，而是通过 DAO 提供的接口获得值对象，修改值对象后通过 DAO 保存到数据源。业务逻辑仅仅通过面向对象的方法操作 DataAccessObject，不必考虑非面向对象的关系数据库操作，也不需考虑访问性能、数据源变更、事务、并发等复杂问题。

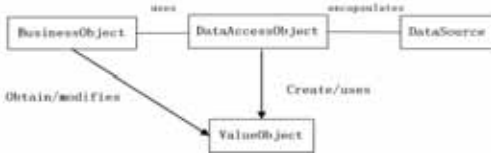

图 1 DAO 模式

DAO 模式完成了一个持久层的部分任务，向业务逻辑开发人员隐藏了对象的持久化细节。不过 DAO 并不是真正意义上的持久层。虽然通过将业务逻辑和数据访问逻辑分离的设计思想，设计人员已经向对象

① 基金项目:浙江省教育厅科研项目(Y200906971);宁波大学科技学院预研项目: (XYY09002)
收稿时间:2009-10-27;收到修改稿时间:2009-12-29

持久化的方向迈进了一大步。但是 DAO 的最大缺陷是没有一个完善的对象－关系映射策略。DAO 模式的所有持久化工作由开发人员编写的 DAO 对象完成，意味着要有专门的开发人员开发和维护一个对象－关系数据库映射。随着应用系统规模的扩大，这种对象－关系映射机制的不完善意味着工作量呈几何上升。如果系统的数据库发生了变更，虽然业务逻辑开发人员不必改动应用逻辑，减轻了变更的困难，但是变更的压力转嫁到 DAO 的开发人员身上，这样的应用系统开发仍然不能从容的面对变更。由于没有良好的映射机制，使得应用开发无法实现组件级的可重用[1]。以上种种情况表明 DAO 模式无法完成构建持久层的重任。

3　持久模型的具体设计与实现

基于 DAO 模式，在设计的时候，我们选择代理模式作为持久层实现模型的框架(如图 2)，意味着持久对象不直接访问关系数据库，而是用 Persistence Manager(持久管理器)作为代理，间接操作数据库。例如，由 Persistence Manager 查找对应的元数据文件找出 Entity(实体)类对应的数据库表，生成对应的 SQL 语句，填充参数交由数据库执行，对于返回结果也是由 Persistence Manager 根据元数据文件找到对应的类构建对象的实例[2]。

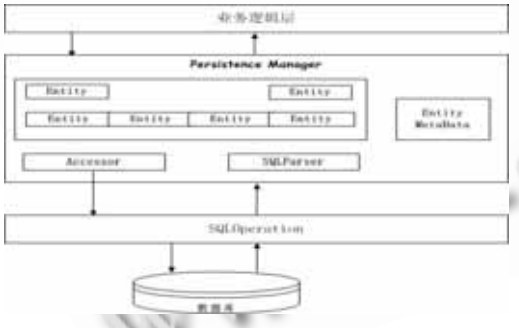

图 2　持久层实现模型的框架

Persistence Manager 定义了一套 API 来完成和底层存储空间的数据传送，这套 API 封装了操作底层存储空间的实现细节。Persistence Manager 不仅可以根据查询生成 Entity 对象，还可以对数据进行批量的新增、修改和删除操作。

Entity 是关系型持久数据在 Java 中的对象表现形式，它映射一个数据库模式中的一个实体定义。例如，一个数据 Bean 类可以映射一个关系型表或视图的定义。Entity 类所能看到的方法都是 PersistenceManager 接口提供的方法。Entity 通过一个工厂类 (SQLpersistenceManagerFactory) 来获一个 PersistenceManager 接口的实例 (SQLPersistenceManager 对象)。任何实现了 PersistenceManager 接口的类都可以替代掉 SQLPersistenceManager 类，并且能够很好的融合到这个持久层中，这也是抽象出 PersistenceManager 接口的一个原因。

SQLParse 是 SQL 语法分析器，根据 Entity 对象提供的方法对应的生成 SQL 语句。

EntityMetaData 是元数据中心，提供对应的 Entity 类的元数据并解析元数据。如要保存一个对象，要找到对象属性相对应的表的字段，以参数的形式发送给 SQLParse，产生相应的 SQL 语句。

Accessor 是关系数据库访问器，由它来具体执行底层数据库的访问操作[3,4]。

3.1　可变机制的设计

我们希望访问底层物理数据的访问器与存储空间的数据形式无关。所以，在这里设计了与底层存储空间无关的访问器 Bean 组件，定义了一个抽象的存储空间操作定义 SQLBase 抽象类，在这个抽象类中定义了与存储空间无关的访问操作，通过运行继承机制，用户就可以开发自己的访问器 Bean 组件。同时针对继承机制所造成的不足，将 SQLBase 的抽象和它的实现部份分别放在独立的类层次结构中。其中一个类层次结构针对访问器接口，另外一个独立的类层次结构针对存储空间相关的操作实现部份，这个类层次结构的根类为 SQLOperation。如图 3 所示。

图 3　SQLBase 抽象和实现分离

SQLBase 的抽象和它的实现之间并没有一个固定的绑定关系，在程序运行过程中，它的实现部份可以

被选择或者切换。SQLBase 的实现可以通过生成子类的方法独立地加以扩充，SQLBase 实现的修改和扩充不会对客户程序造成影响。用户定义的访问 Bean 组件可以共享 SQLBase 的实现。

在图 3 中 SQLOperation 的实现有五个：

SQLQuery：数据库查询操作的实现；

SQLUpdate：数据库更新操作的实现，包括插入、修改、删除；

SQLBatch：数据库批处理操作的实现；

SQLCall：数据库存储过程操作的实现；

SQLXML：XML 文件访问操作的实现。

前四个实现 SQLOperation 子类都是对数据库持久操作的实现，最后一个子类是对基于 XML 文档的存储空间的持久操作的实现。在设计过程中把要创建 SQLOperation 子类对象的工作代理给另一个对象，由它来决定创建什么类型的 SQLOperation 子类对象。这个对象实际上就是一个 factory 对象。SQLBase 仅需向 factory 对象请求一个 SQLOperation 对象，而 factory 对象会返回正确类型的 SQLOperation 对象，这样 SQLBase 类就不会和任何一个 SQLOperation 子类直接耦合[5,6]。

我们使用 SQLBase 中的 setStatement()方法来实现持久操作，setStatement()方法有一个参数 sql，sql 代表的是一个查询语句，如果是对数据库操作，sql 就是一个数据库 SQL 语句，如果是对 XML 文件操作，sql 就是一个 XML 查询语句。SQLBase 向 factory 对象请求一个 SQLOperation 对象时，会将 sql 做为请求参数传递给 factory 对象，factory 对象根据 sql 的类型返回相应类型的 SQLOperation 对象。部分实现代码如下：

```
Class OperationFactory
{
 public SQLOperation getOperation( String sql ) throws Exception
 {
      if( sql 为数据库查询语句 )
           return new SQLQuery();
  //返回查询的 SQLOperation 对象
      else if( sql 为数据库更新语句 )
           return new SQLUpdate();
  //返回更新的 SQLOperation 对象
      else if( sql 为数据库批处理语句 )
           return new SQLBatch();
  //返回批处理的 SQLOperation 对象
      else if( sql 为数据库存储过程调用语句 )
           return new SQLCall();
  //返回存储过程调用的 SQLOperation 对象
      else if( sql 为 XML 查询语句 )
           return new SQLXML();
  //返回 XML 的查询 SQLOperation 对象
      else throw new Exception( "未获支持的操作语句" );
 }
}
Accessor 类的 queryEntity
  public boolean queryEntity( Entity entity, SQLState state ) throws CommonException
 setStatement( state.getSql() ); //设置 SQL 语句
 accessor.executeQuery();   //执行查询
 accessor.first();      //将游标移到结果集的第一条记录
 accessor.get( this );
 //将当前记录的数据设置到 this 代表的 Entity 子类对象
 aceessor.close();      //关闭访问器
}
```

3.2 缓冲机制的设计

在这个系统中 SQLOperation 对象是实际完成和底层存储空间传送数据的对象。对于一个系统运行过程来讲，对象的创建和删除是非常昂贵的，尤其对于像 SQLOperation 这种“重载”对象，开销会特别大。如果客户端请求频繁到达，这时来临时创建和删除 SQLOperation 对象，无论是在空间和时间上开销都会十分大。在具体系统运行过程中，基于性能上以及实现隐藏的考虑，设计缓冲池机制来收集再利用 SQLOperation 对象。当 Persistence Manager 对象发送数据持久请求时，Persistence Manager 将这些请求动态分配给 SQLOpeation 对象，然后由 SQLOpeation 对象完成实际的数据库操作。Persistence Manager 维持 SQLOpeation 对象的缓冲池。

我们把持久操作的抽象和实现分成了两个独立的

类层次结构，在抽象部份和实现部份之间提供一个缓冲机制使访问器 Bean 可以共享实现。我们设计缓冲池时，考虑到 SQLOperation 存在五种实现，如果我们为每个 SQLOperation 的子类创建一个 OperationPool(操作池)的子类，这样便会产生大量的子类，这些子类仅仅在它们所创建的 SQLOperation 对象的类别上有所不同。所以，这里我们采用了对象复合[7](如图 4)。我们让 OperationPool 通过拷贝或者“克隆”一个 SQLOperation 子类的实例来创建新的 SQLOperation 对象，我们称这个实例为一个原型。OperationPool 将它应该克隆和分配的 SQLOperation 对象作为参数，如果所有 SQLOperation 子类都支持一个 newInstance 操作，那么 OperationPool 就可以克隆所有种类的 SQLOperation，因此在我们的缓冲池对象中，用于分配和管理某种 SQLOperation 子类对象的每一个 OperationPool 对象都是一个用不同 SQLOperation 原型进行初始化的 OperationPool 实例。

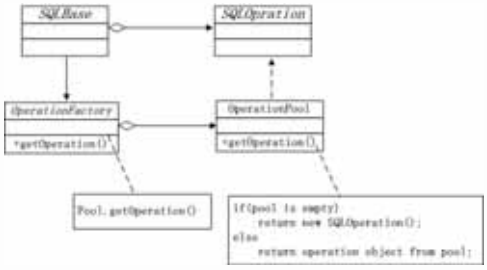

图 4　缓冲池机制

通过原型模式，缓冲池对象独立于 SQLOperation 对象的创建。这样避免了创建一个与 SQLOperation 类层次平行的缓冲池类层次。系统中只需要一个缓冲池类的实现就可以管理所有的 SQLOperation 子类，极大减少了子类的构造。

对于缓冲池回收 SQLOperation 对象。由于用户在执行持久操作的最后一步总是会释放系统资源，即调用 SQLBase 定义的抽象操作 close()，而 SQLBase 本身并不做操作而是直接调用 SQLOperation 对象的 close()方法，所以如果 SQLOperation 对象的 close()方法被调用就意味着缓冲池可以回收该 SQLOperation 对象。为了实现回收机制，我们把 OperationPool 做为观察者，SQLOperation 做为被观察目标，一旦目标（SQLOperation）的 close()方法被调用，目标的观察者（OperationPool）就会得到通知，观察者(OperationPool)得到通知后，就会把目标进行回收，放回到缓冲池中。

4　小结

采用本文模型开发将会在开发效率和运行效率两方面取得明显的效果。以下数据来源与我们项目组开发的多个应用系统的综合分析。

表 1　效率比较图

开发工具	本文模型	EJB	JDBC
开发时间	1	2	6
执行效率	12.05	100	10

由上表可以清楚的看出采用不同的开发模式所需要的开发时间之间的比例关系。采用 EntityBean 的开发时间要比使用传统 JDBC 的开发时间要大大缩短，而采用我们的模型开发比采用 EntityBean 开发还要快。而且我们将数据访问独立出来，对于数据库变更、用户需求变更等软件工程中不可避免的问题，能够更从容的处理。

持久层效率是一个关键的问题，如何优化缓冲机制以及代码重构是我们以后设计的主要考虑方向。随着应用与研究的深入，还会不断出现新的功能需求，因此要不断完善扩展其功能，以最大限度地满足应用要求。

参考文献

1 张绍成,李华林,马玉琴.基于 Java 的对象持久化方法研究.小型微型计算机系统, 2005,26(2):P255－267.
2 史周军,叶晓俊.基于元数据的对象关系映射研究.计算机科学, 2005, 32(5):95－97.
3 张俐.基于 JavaEE 的电信 CRM 数据持久层的实现.计算机工程, 2009,35(6):41－43.
4 苗晓辉.基于 J2EE 的数据持久化的研究与实现.计算机工程, 2007,33(5):272－274.
5 阎宏. Java 与模式.北京:电子工业出版社,2002.
6 Miller J. Design Patterns for Data Persistence [2009-5-3].http://msdn.microsoft.com/en-us/magazine/dd569757.aspx.
7 Ambler SW.The design of a Robust Persistence Layer For Relational Databases. Senior Consultant, Amby softInc[2009-8-2].http://www.ambysoft.com/downloads/ persistenceLayer.pdf, 2008