文章编号:1001-5132（2009）03-0364-06

基于 J2EE 的数据持久模型的设计与实现

孙　霞, 胡旭昶, 姚　畅, 霍　瑞

（宁波大学 科学技术学院, 浙江 宁波 315211）

摘要: 针对面向对象技术和关系数据库企业应用开发的通用环境, 对数据的持久解决方案并实现数据持久模型进行了研究. 基于对象 - 关系映射技术, 使用代理模式进行持久化管理, 将底层的数据访问进行独立封装, 并根据业务请求和映射信息文件动态地生成 SQL 代码, 从而使得对象模型与关系数据库之间具有无关性; 同时, 通过动态分配机制和缓冲机制大大提高了数据访问的效率.

关键词: 面向对象; 关系数据库; 映射; 数据持久

中图分类号: TP311　　文献标识码: A

在 IT 技术飞速发展的今天, 面向对象技术已经成为企业软件开发的主流技术, 利用对象模型能够很好地描述和设计复杂的软件系统. 在软件系统中, 有些对象是需要永久保存的, 而实现对象的存储(数据持久)大多使用的是技术成熟的关系数据库技术. 但是由于面向对象和关系数据库在处理方式上的差异, 导致了关系对象阻抗的不匹配, 使得对象设计者需要花费大量的时间实现对象在关系数据库中的持久. 因此, 良好的数据持久模型可以使开发者能从繁杂的数据持久化工作中解放出来, 将更大的精力放在业务逻辑的获取以及实现上.

1　J2EE 中现有数据持久化技术比较[1]

1.1　Java 对象序列化

对象序列化是最简单的 Java 持久性策略, 是将对象图平面化为字节的线性序列的过程. 然而, 对于复杂的持久化, 对象序列化在很多方面存在缺陷: 它必须要立即存取对象的特征, 同时在序列化的过程中, 需要将对象图从内存具体化到持久存储中, 这就涉及到大量的 I/O 开销; 它没有提供一个从序列化对象图中检索获取数据的查询语句; 在更改 1 个对象的属性时, 如果有错误发生, 它无法实现“回滚”, 因此不适于应用程序对数据完整性的要求, 而且用户不能共享使用数据. 这些缺点的存在使得简单的对象序列化不适合大多数持久化存储的要求.

1.2　JDBC

与简单的对象序列化相比, JDBC 是种复杂的持久对象的技术, 它需要用户手工建立对象 - 关系型数据的映射, 所以开发和维护这样的数据持久层所花费的代价非常庞大. JDBC 可以说是 Java 环境中访问持久层最原始、最直接的方法, 是 Java

收稿日期: 2008-12-25.　　宁波大学学报（理工版）网址: http://3xb.nbu.edu.cn
第一作者: 孙　霞（1977－）, 女, 江苏泰兴人, 硕士/讲师, 主要研究方向: 计算机应用及软件工程. E-mail: sunxia@nbu.edu.cn

中一切数据库操作的基础. 它的优点是运行效率较高，而缺点是实现复杂、编码量大、开发效率低、大项目难以维护. 但无论怎样，使用 JDBC 来直接访问持久数据层仍是当今企业级应用开发中使用最广泛的一种方式.

1.3　EJB 中的实体 Bean

实体 Bean 是 Enterprise JavaBean(EJB)中的持久数据组件，它代表了使用持久化数据的业务过程，是处理底层持久性数据的组件模型. 由于实体 Bean 简化了代码，并能够快速应用开发，所以实体 Bean 的前景十分广阔. 如果告诉 EJB 容器一些关于 Bean 的信息，则容器管理的持久化能够完成所有的数据访问代码，这就大大地减少了 Bean 的代码长度. 由于在 Bean 中没有 JDBC，从而减少了整个应用的开发时间，这也使得代码更容易读懂或者维护. 但是，由于实体 Bean 本身的实现机制导致了它在性能上的缺陷，造成 Entity Bean 容器管理的代价很高，运行效率差，容器不能及时响应数据访问请求.

通过上面的比较分析可知，各种持久化技术都有各自的优缺点. 对于企业信息系统中的数据持久的重要问题，需要选择良好的持久技术来进行数据存储，而数据持久的效率直接影响到应用系统的运行效率. 企业级应用系统对运行效率都有严格要求，采用现有的数据访问模型(如 JDBC、Entity Bean 等)开发企业级的应用系统时，对开发效率和数据访问效率无法兼顾.

2　数据持久化层的总体设计

基于 DAO(Data Access Object)的数据访问模式[2-3]提出了数据持久实现模型——数据持久管理器(DPM). DAO 数据访问模式是将数据访问逻辑从业务逻辑对象中提取出来，并抽象为一个独立的数据访问接口，业务逻辑对象根据接口的操作执行业务逻辑功能，而接口针对使用的数据资源实现为 DAO 对象. 在 DPM 模型中包含数据对象层、持久管理器和数据访问层 3 部分，其体系结构如图 1 所示.

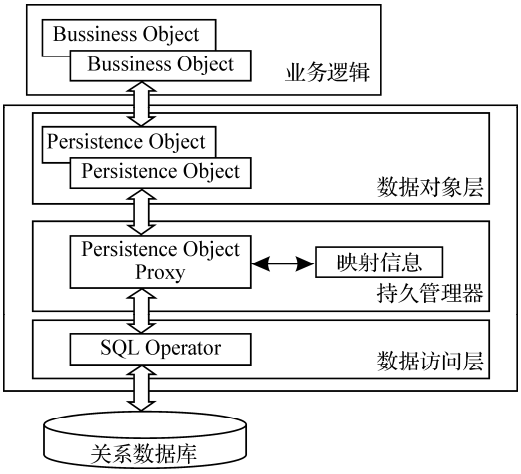

图 1　数据持久层体系结构

数据对象层面向业务逻辑，提供与业务逻辑进行数据交互的接口，从业务逻辑看，此层即是持久化操作的执行者. 持久管理器主要具体实现数据的持久性管理，通过查询对应的映射信息，接受通过数据访问层返回的结果，并将其返回给数据对象层，由数据对象层返回给业务逻辑. 数据访问层是真正与底层数据库进行交互的层，由它来执行数据的添加、存储、删除、查询等操作.

3　持久模型的具体设计与实现

3.1　数据对象层[4-5]

持久对象(Persistence Object)是能够存放在永久性存储空间中的持久数据组件，面向业务层，从业务层看到的就是持久对象类提供的持久化操作. 在该持久模型中，Persistence Object 使用对象－关系映射到 1 个关系型数据库的持久机制，并将自身放入持久存储空间中. 在设计过程中，我们设计了 3 个持久对象类: Entity 类、Relation 类、Rendering 类，如图 2 所示，而它们都继承 SwellBean 类.

SwellBean 类为实现 Swell 接口所有通用 Javabean 的基类，为所有子类打印输出、比较、克隆提

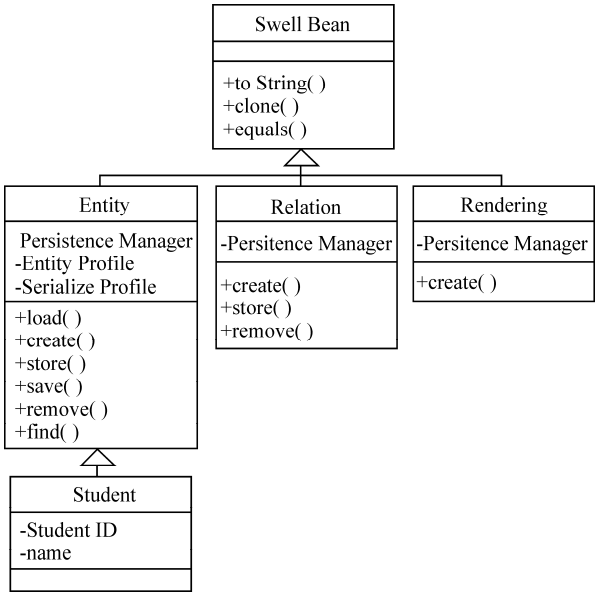

图 2　3 种持久对象类图

供实现.

Entity 类为实体对象的基类，它映射存储空间中的实体定义(如关系数据库中的表和视图). Entity 是关系型持久数据在 Java 中的对象表现形式，对于 Java 对象和数据库之间的信息传送，设计了 1 种机制，用 load( )和 store( ) 2 个专有方法来完成.

Relation 类为关系对象的基类，它映射存储空间实体之间的关系. Relation 对象由多个 Entity 对象聚合而成，多个 Entity 之间有着强烈的相互关系，所以由 Relation 对象统一负责它们的持久化工作.

Rendering 类为透视图对象的基类，它是映射用户显示的持久数据. 内存中的数据并不都是直接从数据库中获得，尤其是诸如统计信息等需要通过逻辑计算才能获得的数据，而 Rendering 对象专门针对这种特殊的需求.

在设计过程中，考虑到性能上的要求, DPM 并没有将数据 Bean 设计为与 EJB 实体 Bean 同样具有隐式持久能力，因为如果由容器来决定什么时候在内存的数据 Bean 和数据库之间来回地传送数据，将会造成非常多的问题. 首先，容器要判断当前实体 Bean 实例所在的事务处理状态；其次，如果有多个实体 Bean 实例代表同个数据，容器就不得不处理多个实体 Bean 实例同步的问题. 大多数 EJB 容器处理的策略是在实体 Bean 实例被钝化时执行 store 操作，在实例被激活的时候执行 load 操作. 这样的结果就会造成很多无谓的操作，即 1 个实例在活动期间，其域数据并没有被更改，但容器仍然会执行 store 操作，这样都会大大降低性能. DPM 的 Entity 被设计为具有显式的持久能力，让开发人员来决定什么时候将数据从底层存储空间装载入内存，什么时候把 Bean 数据保存到底层存储空间，只不过 Entity 向开发人员隐藏了持久的细节，开发人员不需要决定是如何持久的，只需要决定什么时候持久.

以学生信息为例，Student 类从 Entity 类继承，在实例化 Student 类的同时也会实例化 Entity 类，所以在运行期间, 1 个 Student 对象同时隐含地包含了 1 个 Entity 对象.

以客户程序加载学号为“ 001”的学生信息为例的代码片断如下:

```
Student student=new Student();
student.setStudentid(“001”);
student.manangerByPersistenceMananager(pm);
student.load();
```

Entity 类定义的 load 方法的代码片断如下:

```
public boolean load() throws CommonException
{checkManager(“load() #1”);
return_persistenceManager.loadEntity(this);
//转交给持久管理器执行具体的读操作,调用 loadEntity()方法，再将结果集返回给 Entity.
}
```

3.2　持久管理器(Persistence Manager)[6]

在持久模型中，使用代理模式作为持久层实现模型的框架，即 Persistence Ojbect 不直接访问关系数据库，而是用 Persistence Manager 作为代理，间接操作数据库. 也就是说, Entity 类、Relation 类、Redering 类中的 load()、store()等所有方法并不直接访问数据库，而是转交给 Persistence Manager 的

对应方法，由 Persistence Manager 查找对应的元数据文件，找出 Entity 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．同时，Persistence Manager 定义了相应 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，具体静态框架如图 3 所示．图 3 中的 Entity MeteData 放置的是映射信息元数据，通过映射管理器 entityProfileManager 对映射信息进行管理，而 SQLParser 是对应的语法分析器，产生如存储、删除、修改等 SQL 语句，并根据 Entity 对象提供方法对应生成 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.

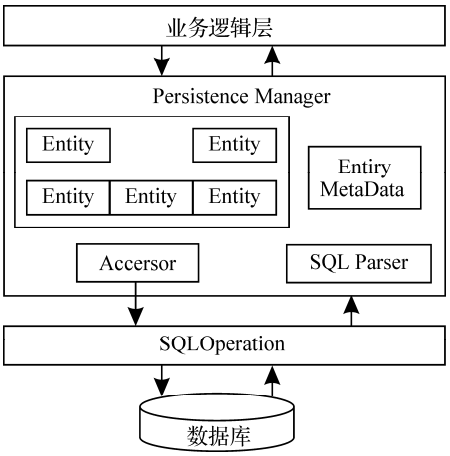

图 3　持久管理器的工作静态结构

底层数据库的访问部分代码片断如下:

```
SqlPersistenceManager 类 //映射信息管理器，是元数据管理中心
_profileManager=profileManager; //得到数据库的连接
_connection=dataSource.getConnection(); // 创建通用访问器
_accessor=new Accessor(_connection); //得到 SQL 语法分析器对象
SQLParser parser=getSQLParser(); //创建 SQL 语句生成器对象
state = new SQLState(); //设置为读操作
state.setName(name); //生成对应的 SQL 语句
state.setSql(parser.forLoad(profile));
public boolean queryEntity (Entity entity, SQL State state) throws CommonException
{……..
    setStatement(state.getSql()); //设置 SQL 语句
    ……
    accessor.executeQuery(); //执行查询
    accessor.first(); //游标移到结果集第一条记录
    accessor.get(this);
    aceessor.close(); //关闭访问器
}
```

3.3　运行时模型

3.3.1　数据访问对象缓冲机制[7]

在具体系统实现时, SQLOperation 对象是实际完成和底层存储空间传送数据的对象，对于 1 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，如果客户端请求频繁到达，临时创建和删除 SQL-Operation 对象无论是在空间和时间上开销都会十分巨大. 因此，基于性能上以及实现隐藏的考虑把它独立出来，并设计缓冲池机制来收集及再利用 SQLOperation 对象，相应机制如图 4 所示.

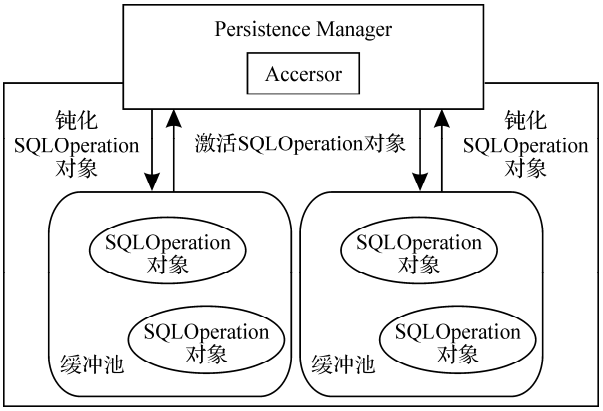

图 4　SQLOperation 对象缓冲池

3.3.2 数据访问对象动态分配机制[8]

从面向对象设计技术来看, SQLOperation 类是 Persistence Manager 类的组件，但是这样就和某种底层存储空间传送数据的实现细节绑定在一起，如果对这个抽象的实现细节进行修改，就会对所有 Accessor 类产生影响，而且如果要处理多种底层存储空间, SQLOperation 类就非常难以进行扩展. 而把 SQLOperation 类独立，则可为 Accessor 类提供多种实现，其关系如图 5 所示. SQLOperation 对象运行期间，动态地绑定到 Accessor 对象中，这样 Accessor 对象完全不用修改就可以拥有和多种底层存储空间传送数据的能力，比如可以不使用数据库，而把数据保存到 1 份 XML 文件中，此时只要用户提供一个基于 XML 文件的 SQLOperation 实现即可，但其 Accessor 类完全不用修改.

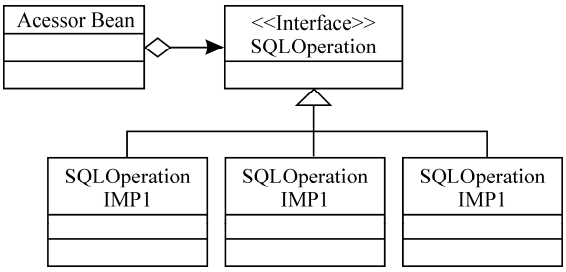

图 5 SQLOperation 多重实现

在运行过程中，由 SQLOperation 对象的工厂类来负责创建具体的 SQLOperation 对象. 当 Entity 或 Accessor 对象要进行数据持久操作时，它们先向 SQLOperationFactory 对象请求 1 个 SQLOperation 对象, SQLOperationFactory 对象会根据请求的类型从相应的 SQLOperation 对象缓冲池激活 1 个 SQLOperation 对象，如果缓冲池中已经没有可用的 SQLOperation 对象，SQLOperationFactory 对象就会自己创建 1 个 SQLOperation 对象并激活它，然后 SQLOperationFactory 对象将这个 SQLOperation 对象的引用返回给请求者. SQLOperationFactory 对象还会监视 SQLOperation 对象的状态，如果请求者使用完 SQLOperation 对象, SQLOperationFactory 对象就会将这个 SQLOperation 对象进行钝化，然后将其放回到相应的缓冲池中，其代码片段如下:

```
Class SQLOperationFactory
{   public SQLOperation getOperation(String sql ) throws Exception
    {
        if(sql 为数据库查询语句)
            return new SQLQuery();
        else if(sql 为数据库更新语句)
            return new SQLUpdate();
        ……..
        else if(sql 为 XML 查询语句)
            return new SQLXML();
        else throw new Exception(“未获支持的操作语句”);
    }
}
```

从上面的描述可以看出，运行时的 Persistence Manager 由如下几个部分构成: (1)SQLOperation: 具体底层存储空间操作的实现者; (2)缓冲池: SQLOperation 对象在内存中的集中共享空间，不同类型 SQLOperation 对象拥有不同的缓冲池; (3)SQLOperationFactory: SQLOperation 对象的管理者，它负责 SQLOperation 对象的创建、删除、分配以及回收.

4 小结

数据访问层是应用系统的基础，它的效率直接影响到整个应用系统的运行效率. 通过设计并实现了一个基于 J2EE 架构的功能相对完备的持久层，可减轻开发人员在程序设计时对数据库具体设计方案的考虑程度，从而使整个开发过程不被数据库及其变化所限制. 但是，在具体的企业应用环境中会遇到更多不同的问题，而笔者的模型解决了在项目中遇到的普遍问题. 目前在许多方面的实现都采用了简化原则，在功能上有较大的扩

充余地, 随着应用与研究的深入, 还会不断出现新的功能需求, 因此要不断完善其功能, 以最大限度地满足应用要求. 同时, 持久层的效率是个很关键的问题, 特别是 SQLOperator, 所有的数据库访问都由它来具体执行, 如何优化模型如设置缓冲池、重构代码、简化管理是主要考虑方向.

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[J]. 小型微型计算机系统, 2005, 26(2):255-267.

[2] Clifton Nock. 数据访问模式: 面向对象应用中的数据库交互[M]. 鄢爱兰, 译. 北京: 中国电力出版社, 2004.

[3] Erich Gamma, Richard Helm, Ralph Johson, et al. Design patterns: Elements of reusable object-oriented software[J]. New York: Addison Wesley, 1995.

[4] 杨善林, 王毅, 马溪骏. 面向对象的数据持久层解决方案——Java 数据对象[J]. 计算机应用研究, 2005, 22(11):28-30.

[5] 史周军, 叶晓俊. 基于元数据的对象关系映射研究[J]. 计算机科学, 2005, 32(5):95-97.

[6] Scott W A. The design of a robust persistence layer for relational databases[EB/OL]. [2000-11-07]. http://www.pdbm.de/extern/persistenceLayer.pdf.

[7] Patrick Chan. Java developers almanac 1.4 volume 1 examples and quick reference[M]. New York: Pearson Education Inc, 2002.

[8] Jushua Bloch. Effective Java programming language guicle[M]. New York: Pearson Education Inc, 2003.

Design and Implementation of Data Persistence Layer Model Based on J2EE

SUN Xia, HU Xu-chang, YAO Chang, HUO Rui

(College of Science and Technology, Ningbo University, Ningbo 315211, China)

Abstract: Aiming at developing object-oriented technology and relational database in a general environment for enterprise applications, this article studies the approaches for data persistence solution and model construction. Proxy model is used to manage data persistence based on O/R Mapping. The encapsulated underlying data are made completely accessible, with the object model and relational database being independent of each other by building SQL code dynamically with the query of operation and mapping information. The data accessing efficiency is greatly uplifted through the dynamic allocation and the buffering mechanism.

Key words: object-oriented; relational database; mapping; data persistence

CLC number: TP311　　**Document code:** A

（责任编辑　章践立）