文章编号:1006-2475(2006)06-055-03

采用ORM技术的软件开发方法研究

陈春玲,朱常宝,严 劲

(南京邮电大学计算机科学与技术系,江苏 南京 210003)

摘要:首先介绍对象模型与关系模型的两个基本定义并推出两者之间的映射函数,然后通过UML建立对象模型与关系模型的映射,最后介绍一种实现关系数据库与对象之间自动映射的ORM技术,说明采用ORM框架的软件开发总体设计和关键技术。

关键词:对象关系映射;对象模型;关系模型;统一建模语言;透明持久对象层

中图分类号:TP311 **文献标识码**:A

Research on Software Design Method Using ORM Technique

CHEN Chun-ling, ZHU Chang-bao, YAN Jing

(Computer Science and Technology Department, Nanjing University of Post and Telecommunication, Nanjing 210003, China)

Abstract: This paper first proposes the basic definitions of object model and relation model and their mapping function, then constitutes the mapping using UML, at last, proposes a kind of technology which implements auto mapping between object model and relation model, and introduces the general software design and pivotal techniques using ORM(object relation mapping) structure.

Key words: ORM; object model; relation model; UML; transparent persistent layer

0 引 言

当前,在大多数采用OO方法且涉及数据库的软件开发中都需要设计对象和关系数据库,在软件的业务逻辑层和用户界面层要对编写的对象进行操作,而在改变对象的信息后还需要把对象的信息保存在关系数据库中。因此,在开发一个业务逻辑层和用户界面层分离的应用程序时,程序员要写不少数据访问层(Data Access Layer, DAL)的代码,用来从数据库读取、保存、删除对象信息以及改变对象状态等任务。通常,这些数据访问代码基本类似,不管是早期的嵌入式SQL还是后来的JDBC/ODBC方式,都是先传入操作对象,然后设置存储过程,再设置属性与对象对应,最后执行存储过程。

这种在每个项目中都重复且具有相同模式的代码明显是一种资源与人力的浪费,在这种情况下对象关系映射(Object-Relation Mapping, ORM)技术应运而生。它是在关系型数据库和对象之间产生一个自动映射,这样在具体的数据库操作中就不需要再与复杂的SQL语句打交道。软件设计人员只需要关注业务逻辑中的对象架构,而不是底层的重复性的数据库SQL和JDBC代码。据统计,采用ORM可将软件开发时间和成本缩短40%,并且由于很方便地将业务层与实际的数据存储分开,从而极大地提高数据可读写性,也简化了代码的调优与测试。很多开发人员和专家都预测,ORM将成为面向对象开发中的主流开发模式。

1 对象模型与关系模型

《UML参考手册》[Rumbargh 第1版]中将对象定义为"具有良好定义的封装了状态和行为的边界的具体实例,类的实例",从数理的角度也可以将对象视为数据和函数的内聚捆绑。关系模型以集合论中的关系作为数学基础,它是一个(数)域的笛卡尔积的子

收稿日期:2005-08-10

作者简介:陈春玲(1964-),男,江西南昌人,南京邮电大学计算机科学与技术系副教授,研究方向:计算机软件与理论,计算机软件在通信中的应用;朱常宝(1978-),男,江苏洪泽人,硕士研究生,研究方向:计算机软件与理论;严劲(1977-),男,安徽合肥人,硕士研究生,研究方向:计算机软件与理论。

集。于是我们引出对象模型中的对象类和关系模型中的关系的一般定义。

定义1 一个对象类可简单定义为 $O(A_1,\ldots,A_n,M_1,\ldots,M_k)$，其中，$A_i(i=1,\ldots,n)$为对象类的属性，定义在域 Di 上；$M_i(i=1,\ldots,k)$为对象类的方法，其返回值定义在域 Di 上。

定义2 关系 R 是笛卡尔积 $D_1\times D_2\times\ldots\times D_n$ 的一个子集，$D_1\times D_2\times\ldots\times D_n$ 是所有的有序的 n 元组 $<d_1,d_2,\ldots,d_n>$ 的集合，其中 $d_1\in D_1,d_2\in D_2,\ldots,d_n\in D_n$，$D_1\times D_2\times\ldots\times D_n$ 称为关系 R 的域。关系模式的一般形式可表示为 $R=<U,D,DOM,\sum>$，其中 U 为组成关系 R 的全部属性的集合，D 为域的集合，DOM 为 U 与 D 间的映射，$\sum$为 U 上的一组依赖约束。

从定义不难看出，就单个的对象和关系而言，对象的属性及关系元组均定义在域 $D_i(1<=i<=n)$ 上；而对象与关系分别是对物体类型抽象（Abstract Data Type，ADT）与数学抽象，对象间和关系（或者表）间的联系也不外乎 1:1、1:n、m:n 三种。所以，从单纯数理角度，可以定义对象与关系的如下映射。

定义3 映射函数 F 是满足下列条件的函数：

F 的定义域为对象所有实例的集合，设为 I；F 的值域为关系数据库中所有视图的幕集，设为 $\prod$；对一个对象实例 $o\in I$，其所有方法与属性的集合为 S。$V\in\prod$，即 V 为若干视图的集合。可以有如下结论成立：①对任意 $s\in S$，必存在唯一的 $v\in V$，即对某一对象的任一属性和方法，存在唯一视图，该视图包含该属性；②对任意 $o\in I$，S 为其对应的方法与属性集合，可通过其对应的原始属性表通过有限次的无损失连接性及保持函数依赖性的分解，最后得到的关系模式满足 BCNF 范式。

结论①通过映射函数的定义是显然成立的，而结论②成立是因为可以先将某一对象实例的所有属性列于一张关系表中，并且已有理论证明任一关系模式可找到一个分解达到 BCNF 且具有无损失连接性。

虽然从数学上很容易将对象模型与关系模型有机地统一起来，但是关系理论的基础之一是数据和使用数据的程序能够而且应该是相互独立的，这与对象技术的整个理念是不一致的。对象技术鼓励设计者使用对象而不是表来思考数据，对象和使用对象的方法是不可能彼此分开的。在关系理论中，数据应该被组织成规范的表，也就是数据应该按唯一的方式组织，使得程序员能够消除冗余，确保数据变化的一致性。这种设计技术的引入确保了关系表中的数据是一组独立的、通过键相关的数据。这种技术来自集合论的数学理论，但问题是集合论不能表达数据之间所有的关系和结构。

面向对象设计提供了功能强大且便于理解的建模工具 UML，通过 UML 可以将对象与关系间映射比较方便直观地表示出来。

2 通过 UML 建立对象模型与关系模型映射

采用 OO 方法时，对象间两两关系不外乎一对一、一对多、多对多的情况，同时要考虑采用 OO 方法时继承、封装和多态的特殊特性，下面通过统一建模语言（uniform model language，UML）建立对象间 E-R 模型的同时将其映射成相应的关系模型。

对象间的对应关系为 1 : 1 时，关系表间的映射可设置任一表的主键为另一张表的外键即可。如图 1 中，Item 和 ItemPicture 两个对象关系为 1:1，映射关系表可设置表 Item 的主键 item_id 为表 ItemPicture 的外键。

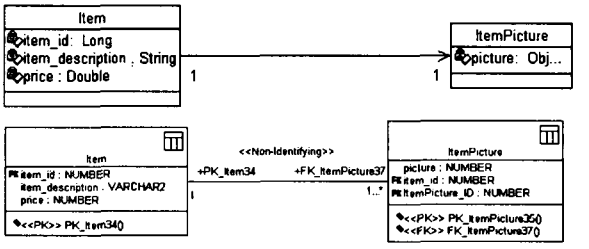

图1 1:1 映射

对象间的对应关系为 1 : n 时，关系表间的映射可将“1”端的主键设置为“n”端的外键。如图 2 中，对象 Item 与 Ordereditem 为 1:n 关系，映射关系表可设置表 Item 的主键 item_id 为表 Ordereditem 的外键。

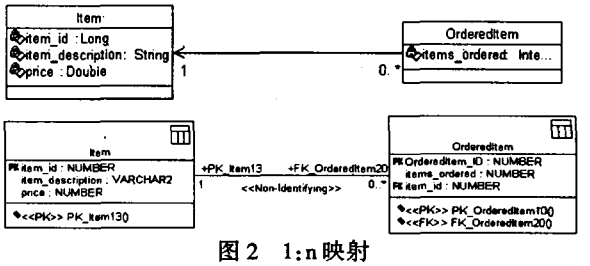

图2 1:n 映射

对象间的对应关系为 m : n 时，关系表间的映射可将两个表的主键提取列入一张单独的表，并将这两个主键分别作为新生成表的主键和外键。如图 3，对象 Customer 和 Employee 为 m:n 的关系，映射关系表可分别从表 Customer 和 Emloyee 中提取主键 Customer_ID 和 Employee_ID 构成新表将这两个表关联起来。

对象间存在继承关系时，关系表间的映射可将父对象对应表的主键设置为子对象对应表的主键和外键，图

4 表示子对象与父对象之间是"is a kind of"的关系。

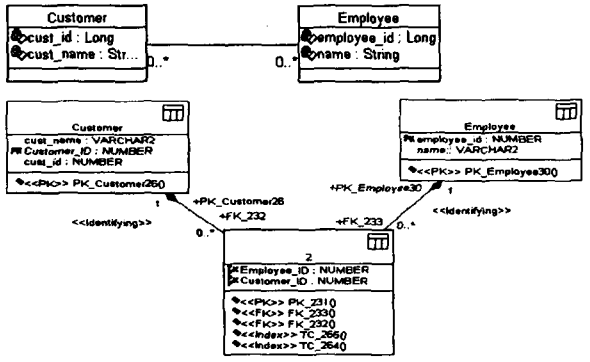

图 3　m:n 映射

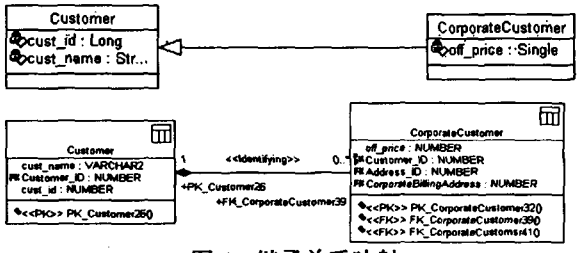

图 4　继承关系映射

以上所列的四种是面向对象程序设计中最常见最基本的数据模型,在实际的设计过程中还会有许多特殊情况,比如在将对象映射到一张表中由于有属性值的重复可能会有数据冗余,而为了减少这种冗余就要通过关系模式的分解将一张关系表分解成多张没有函数依赖的关系表,此时会出现一个对象对应多张关系表;有时一个父类对应多个子类,而如果每个子类相对于父类变化的属性很少或只是利用多态性对父类的方法进行重载而并没有增加属性,此时为了节省空间,可设置多个子类只对应于一张父类表;还有对象多重继承时可能会涉及表间设置多个主键和外键(见图 5)等等。这些都可在上述四种基本映射的基础上利用关系模式的方法加以解决。

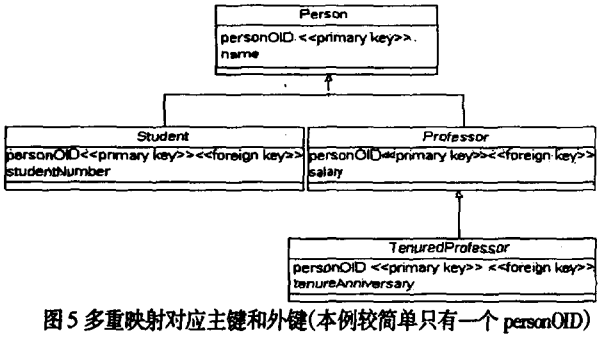

图 5 多重映射对应主键和外键(本例较简单只有一个 personOID)

3　采用 ORM 框架的一般方法及关键技术

采用 ORM 框架的软件设计要求先进行对象建模而不是数据建模,即首先从问题领域出发,以面向对象(或面向方面)的思路来分解问题,然后把这种对象模型映射到数据库,而实现对象与关系的映射一般可通过组件技术来实现,比如 Java 开源项目中比较流行的 Hibernate、JDO(Java Data Objects)、Spring 等架构技术,均是将对象持久化到关系数据库中的实现技术。对特定的数据存取这一点 ORM 与面向对象数据库(OODBMS, Object Oriented DataBase Management System)有所区别,OODBMS 试图直接在数据库中提供内存对象模型,存储对象模型,遍历对象模型,检索对象模型从而造成速度上的问题,而 ORM 中对象模型本身是最适合内存中维护的,比如将引用从一个对象实例变成另外一个对象实例。在内存中维护对象模型,最后以事务的形式千变万化以后最终的对象关系快照让 RDBMS 来持久。

ORM 框架涉及的关键技术主要有:

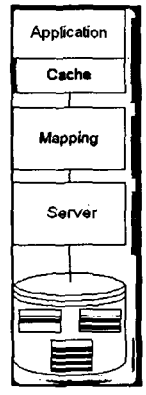

图 6　透明的持久对象层的一般架构

1. 透明的持久对象层(Transparent Persistent Layer)。

采用 ORM 方法要求在软件设计中首先要分析问题域,找出所有问题域中相关事物,从中抽象出对象;然后从抽象出的对象中找出所有的持久对象(PO, Persistent Object),所谓持久对象就是由数据库管理系统负责管理的,可以永久保留,将来可被提取的对象,将这些持久对象以"一类一表格"的原则映射到数据库中,通过数据库管理系统建立表格;然后定义持久对象,所有持久对象均采用"双构造函数"的方法在程序中进行构造,其中,一个构造函数参数为所有初始化该持久对象的值,封装数据模块中"存"操作,另一个构造函数的参数为唯一标识该持久对象的值,封装数据模块中"取"操作。其它数据库操作由持久对象相应方法封装,所有与数据库层发生交互的动作,均放在专门的数据模块中,由中间件层持久对象相应方法封装,做到数据与界面的分离。最后就是定义其它非持久对象,具体软件功能由相应对象协同实现。图 6 所示即为透明的持久对象层的一般架构,Mapping 中间件层透明地实现了持久化对象与数据库表的映射。

2. 客户端应用对象空间与数据库服务器端元组空间的映射。

OO 方法涉及到对象空间的概念,所谓对象空间(object space)是指所有基于一定模式的对象的集合,理想的对象空间是相互独立而没有关联的,每个应用程序可以包括所属对象空间所有对象且相互独立。但在 C/S 或 B/S 模式中,真正的对象空间位于服务器中,如果仍然要求每个应用客户程序独占对象空间就会造成在一个应用程序访问服务器时与其它应用程序只能互斥进行。通常的解决办法是每个客户端应用程序所占有的对象空间作为服务器对象空间的副本,而整个客户端应用对象空间的集合即构成整个或部分的服务器对象空间。

(下转第 60 页)

进一步分析我们发现，ClientRental关系中有三个候选码，三个主属性，但是主属性之间并不存在着局部的或传递的函数依赖，其他三个关系Client、Property、Owner中都是单属性构成的候选码，此时关系模式符合BCNF的要求。

5 规范化处理的原则

泛关系模式的规范化处理通过对模式的分解来实现。一般地，关系模式R相对于函数依赖集F分解成数据库模式 $\rho=\{R_1,R_2,\cdots,R_k\}$，应具有三个特性：①$\rho$是BCNF模式集或3NF模式集，②无损连接分解，③保持函数依赖集F。其中①使分解后的关系模式概念趋向单一，以清除冗余和操作异常，②和③可以用来衡量模式分解前后关系模式是否保持数据等价及语义等价[2]。

定义5.1 设R(U)是一个属性集U上的关系模式，F是R上的一个FD集。R分解成数据库模式 $\rho=\{R_1,R_2,\cdots,R_k\}$，如果对R中满足F的每一个关系r，都有

$$r=\pi_{R_1}(r)\bowtie\pi_{R_2}(r)\bowtie\cdots\bowtie\pi_{R_k}(r)$$

那么称分解ρ相对于F是“无损连接分解”(Lossless Join Decomposition)，简称为“无损分解”，否则称为“损失分解”(Lossy Decomposition)。

其中 $\pi_{R_i}(r)$ 表示关系r在模式 R_i 属性上的投影。

定义5.2 设 $\rho=\{R_1,R_2,\cdots,R_k\}$ 是R的一个分解，F是R上的FD集，如果有 $\bigcup\pi_{R_i}(F)=F$，那么称分解ρ保持函数依赖集F。

算法5.1 无损分解的测试[3]

(1)构造一张k行n列的表格，每列对应一个属性 $A_j(1\leqslant j\leqslant n)$，每行对应一个模式 $R_i(1\leqslant i\leqslant k)$。如果 A_j 在 R_i 中，那么在表格的第i行第j列处填上符号 a_j，否则填上 b_{ij}。

(2)把表格看成模式R的一个关系，反复检查F中每个FD在表格中是否成立，若不成立，则修改表格中的值。修改方法如下：对于F中一个FD X→Y，如果表格中有两行在X值上相等，在Y值上不相等，那么把这两行在Y值上也改成相等的值。如果Y值中有一个是 a_j，那么另一个也改成 a_j；如果没有 a_j，那么用其中一个 b_{ij} 替换另一个值(尽量把下标ij改成较小的数)。一直到表格不能修改为止。(这个过程称为chase过程)

(3)若修改的最后一张表格中有一行是全a，即 $a_1a_2\ldots a_n$，那么称ρ相对于F是无损分解，否则称损失分解。

测试一个模式分解是否保持FD，可以逐步验证F中的每个FD是否满足 $\bigcup\pi_{R_i}(F)=F$。

对于上例，使用算法进行测试，可以发现满足无损分解和保持函数依赖集的特性。

参考文献：

[1] 施伯乐，丁宝康，汪卫.数据库系统教程(第2版)[M].北京：高等教育出版社，2003.

[2] Thomas Connolly, Carolyn Begg. Database Systems: A Practical Approach to Design, Implementation, and Management (Third Edition) [M]. 北京：电子工业出版社，2003.

[3] 萨师煊，王珊.数据库系统概论(第3版)[M].北京：高等教育出版社，2000.

(上接第57页)

ORM技术最广泛地应用于基于C/S或B/S模式的应用开发中，因为关系数据库一般是作为后台服务器而独立于前台应用进程的。由于客户端的对象空间之间一般不可能是独立的子集，这就要求应用程序并发地进行数据库的存取、修改时考虑到同步。一种做法是在一个应用程序对数据库进行写操作时进行锁定，其它的应用程序只能读或等待，这种做法类似于应用程序独占对象空间。另一种做法是应用程序只有在作为副本的用户对象空间从复制起并没有做任何改动的情况下才能对数据库进行修改。后一种是比较理想的锁定，但难以实现，一种可行的实现方案是采用如图6的应用缓存机制，同时设定并记录下服务器对象从被复制后的状态标志位，只有当其标志位未作修改时才能进行修改操作。

参考文献：

[1] Mark L Fussell. Foundations of Object Relational Mapping [DB/OL]. http://www.chimu.com, 1997-07-27.

[2] IBM Corporation. Rational Software White Paper : Mapping Object to Data Models with the UML [Z]. Rational the E-development Company, 2001-07.

[3] [美]Michael Blaba, Wiliam Premerlani. 面向对象的建模与设计在数据库中的应用[M].宋今，赵丰年译.北京：北京理工大学出版社，2001.

[4] 萨师煊，等.数据库系统概论(第3版)[M]. 北京：高等教育出版社，2003.

[5] C Herbert. An Introduction to Relational Algebra (Computer Information Systems 205 Database Management Systems of College of Philadelphia) [DB/OL]. http://www.ccp.cc.pa.us/, 2003-02-26.

[6] Hibernate Corporation. Hibernate 2 and Hibernate 3 Documents [DB/OL]. http://www.hibernate.org, 2004-04-07.

[7] Erich Gamma, Richard Helm, Ralph Johnson. 设计模式：可复用面向对象软件的基础[M].李英军，马晓星，蔡敏，等译.北京：机械工业出版社，2000.