利用数据库处理多个对象间的关系

陈文宇*1 许鸿川2

(1. 电子科技大学计算机科学与工程学院 成都 610054; 2. 成都电子机械高等专科学校 成都 610031)

【摘要】在面向对象的设计中，针对对象实体的保存和管理(增加、修改、删除、查询)较困难的问题，根据数据库管理系统的优点，提出了将数据库与对象的属性联系起来，把数据库运用到面向对象的设计中，解决从对象到数据库的存贮、从数据库到对象的恢复两个问题，再利用数据库处理面向对象的关系解决面向对象设计中类之间的多种关系和继承等独特的特性。该方法简便、实用。

关 键 词 数据库管理; 面向对象设计; 类之间关系; 继承

中图分类号 TP311

Application of Data Base Management System Used in Object_Oriented Design

Chen Wenyu[1] Xu Hongchuan[2]

(1. College of Computer Science amd Engineering, UEST of China Chengdu 610054;

2. Chengdu Eletromechanical College Chengdu 610031)

Abstract For the reason of the difficult to save and manage lots of realities of objects in object_oriented design, such as add，modify，delete，query，based on the advantage of data base management system. The relation of data base management and object attribute，so we can put data base management into oriented object design，and we need to solute two problem: the store of object to data base and data base to object. Because oriented object has some character，such as classes relation and heirship，so we also need to solute them. The method is easy and useful.

Key words data base management; object oriented design; classes relation; inherit

1 类与对象实体集合的关系

类是具有相同属性和服务的一组对象的抽象定义，它为属于该类的全部对象提供了统一的描述，给出了属于该类对象的抽象定义，而对象则是符合这种定义的实体。

少量的对象实体处理较简单，当系统中某个类的对象实体数量很多时，管理(增加、修改、删除、查询)这些实体是系统必须解决的问题。所以，系统建立的对象模型，必须为管理已有对象实体而建，它可用于增加、修改、删除、查询已有对象的实体，称为实体管理模型。实体管理模型在面向对象的分析设计中也是一个类，这个类在建模时容易被忽略。在纯粹的面向对象的程序设计中，一般使用一个管理对象的类来进行多个对象的管理[1]。

本文用一个人通信录的处理实例，建立一个传统实体管理模型。通信录是一个对象，它所包含

2001年12月26日收稿

* 男 32岁 硕士 讲师

的每一个其他人的情况也是一个对象。

```
class    OBJclass
    {
      private:
            char    * name;
            int    age;
            int    sex;
            char    IdNo[18];
            char    homephone[12];
            …
      public:    //对以上属性的操作；
          …
    };
```

OBJclass **的对象是一个通信录里的人，对整个通信录的管理，不能直接放在该类中实现，而应该是通信录类提供的操作。**

```
  class    ManageOBJclass
    {
      private:
            OBJclass    *    OBJS;                    // 通信录中的对象数组，对象的包含关系
            int    size;                               // 通信录中包含人的个数
      public:
            ManageOBJclass(int numb)
          {     OBJS=new    OBJclass[size=numb];      }
          ~ ManageOBJclass()    {      delete [ ] OBJS;        }
            void    append(OBJclass    oneobj);          // 将一个人添加到通信录中
            OBJclass    findone(char    * onename);      // 根据姓名查找一个人
            OBJclass    findone(char    * onehomephone);  // 根据家中电话查找一个人
            void    deleteone(char    * onename);        // 根据姓名删除一个人
            void    modifyeone(char    * onename);       // 根据姓名修改一个人
            void    expend(offset);                      // 增加通信录的容量
            …
          };
```

该程序运行结束后，通信录中所有人的情况没有被保存，并且对象的增加、修改、删除、查询功能的实现较麻烦，也不全面，而数据的永久保存、增加、修改、删除、查询功能则是数据库的强项，因此，实体管理模型对应着数据库。

2　数据库是对象属性的永久保留形式

数据库可以用来保存系统中已实例化的一些对象特征，如属性。但数据库不能直接将整个对象存贮起来(至少不能直接存贮对象的服务)，即数据库不能直接作为永久对象的保留媒介，解决此问题有以下两种思路：

1) **以数据库作为出发点**

把对象、类等不能直接保留在数据库中的部分(如服务)，以数据库的概念及方法间接地保留在

数据库中，再从数据库中取出后还原。由此建立了属性-方法的对应关系，每一个独立的联合记录就构成了一个完整的对象。

2) 以面向对象考虑

数据库仅记录对象的属性，完成属性的增加、修改、查找、删除功能，至于对象中的“服务”等，不放入数据库中，而在面向对象的分析及设计以至最终的实现中解决，故数据库仅是永久对象属性的保留，而不是整个永久对象的保留。

第一种思路破坏了面向对象概念的完整性，保留在数据库中间接部分及其转换的过程在面向对象的分析及设计时的建模，无论在设计还是实现上，都存在很大的复杂性，而第二种思路实现则非常容易，可以充分发挥数据库与面向对象语言的长处，且绝大多数的OO语言都支持。基于第二种思路，将数据库与对象的属性联系起来必须解决从对象到数据库的存贮及从数据库到对象的恢复，解决的方法如下：

1) 从对象到数据库的存贮

对象属性的改变方式可以划分为录入性的改变和程序触发性两种，无论哪种方式都必须把这个改变永久性地记录到数据库中。对于录入性的改变，直接设计一个录入界面并将录入的属性存盘即可。对于程序触发性的属性改变，在属性所在的类模型(类)中设计改变这些属性的服务。

2) 从数据库到对象的恢复

在永久属性所在的模型中设计“取出”服务，该服务首先根据属性所在的类，实例化出一个空对象(所有属性值均为初始值的对象)，再根据取出的条件，从数据库中取出符合条件的对象属性值，并将具体的属性值赋予该实例的对应属性，最终返回对象实体。当“取出”服务根据属性所在的类实例化出一个对象时，这个空对象已经具有了该类所有的一切服务。

3 在数据库中体现类属性的相互关系

数据库中只记录对象的属性值，不处理对象的服务，对象的服务是面向对象的分析、设计以至最终实现中解决的问题。因此，类的关系中数据库只需解决属性的关系。

数据库是对象属性的永久保留，而对象所在的类可能存在继承关系，属性是从其他类继承而来的，因此，类的继承关系必须在数据库中得到体现。

类之间不仅仅存在继承关系，还有其他的关系。同样，所有关系在数据库设计中都必须得到体现。不考虑这些关系，将破坏面向对象分析设计或数据库理论的完整性、统一性，不能正确将面向对象的分析设计与数据库理论真正结合起来。

3.1 类与类之间的关系

类与类之间的关系可以分为三种：整体-部分结构、一般-特殊结构和实例连接关系。

1) 整体-部分结构

整体-部分结构的定义如下：如果对象*A*是对象*B*的一个组成部分，则称*B*为*A*的整体对象，*A*为*B*的部分对象，并把*B*和*A*之间的关系称作整体-部分关系。整体-部分结构表现为“a part of”的关系，如发动机与轿车、电脑与CPU等。

2) 一般-特殊结构

一般-特殊结构的定义如下：如果类*A*具有类*B*的全部属性和全部服务，而且具有自己特有的某些属性或服务，则*A*称为*B*的特殊类，*B*称*A*的一般类。一般-特殊结构表现为“is_a”或“a kind of”的关系，即继承关系，如蝴蝶与昆虫、卡车与轿车、人与学生等。

一般-特殊结构使特殊类通过继承而拥有一般类的特征，整体-部分结构使整体对象通过组装而拥有部分对象的特征。尽管途径不同，但结果却一致，即一些对象拥有另一些对象的特征。

3) 实例连接

实例连接用于表达对象之间的静态联系，是指最终可通过对象属性来表示的一个对象对另一个对象的依赖关系。如上级与下级间、教师与学生的授课关系等。

3.2 类之间三种关系的实现方式

1) 整体-部分结构

整体-部分结构有两种实现方式：一是用部分对象的类作为数据类型，静态地说明整体对象中代表部分对象的属性变量，这样部分对象就被嵌入到整体对象的属性空间中，形成嵌套对象；二是把整体对象中的属性变量定义成指向部分对象的指针或部分对象的对象标识，运行时动态创建部分对象，并使整体对象中的指针或对象标识指向它[2, 3]。

2) 一般-特殊结构

利用OO语言的继承实现。或者将一般-特殊结构在建模时转化为整体-部分结构，用整体-部分结构的方法实现。

3) 实例连接

实例连接的概念比较难以理解，尤其是复杂的实例连接。如用户使用某个文件的关系，可能要求附加表明“优先级”、“使用权限”等与这个关系密切相关的其他信息。

本文引进一种新的概念来解决这个问题，称为“作为类的关联”，关联是一种类，既有属性，又有操作，链(即实例连接)是关联的对象实体。对象不仅可以用于表示有形的事物(如用户、工作站)，也可以用于表示无形的事物(如使用权)。当两类对象之间的实例连接比较复杂时(带有一些属性或操作)，说明在它们之间存在某种尚未用对象加以描述的事物。在上面的例子中，用户与文件之间在建模时应建立一个“使用权”类，该类的属性有“优先级”等，每种使用权可以给多个用户，也可以给多个文件。故只要解决了以上三种关系，将数据库理论应用于面向对象的分析、设计与实现则完全可能。

3.3 将父类的属性组装到子类中实现继承

将一般-特殊结构与整体-部分结构是实现“一些对象可以拥有另外一些对象的特性”的两种方式的理论加以推广，即将父类的属性组装到子类中就有理论根据。

在面向对象的分析与设计中，数据库仅是实现永久属性保存的一种辅助。因此，在分析阶段，无需转化为数据库可以实现的整体-部分关系。在数据库的设计阶段，将具有继承关系、而又需要永久保留的属性考虑用组装的方法。

组装父类、子类属性的方法可以分为紧密组装与连接组装。紧密组装是指父类的属性直接与子类的属性组装在一起，共同形成子类的数据库结构。连接组装是父类的属性与子类的属性并不在同一个实际数据库中，它们各自所在的数据库通过某个字段相联系，如数据库理论中常见的Master-Detial结构。

如果父类的已有对象实体已经用数据库管理，尤其是子类的对象实体可能是父类已有实体的某些特性扩充时，子类与父类间应该用连接组装。

按照连接组装的概念，在数据库设计时，除了要在子类的属性数据库中有对象实体的唯一标识字段外，还要有标定该对象实体与父类属性数据库中相应对象实体相连接的字段。

3.4 对象的恢复

在子类的“取出”服务中，首先实例化出一个子类空对象(所有属性值均为初始值的对象)，此时，这一实例化的子类空对象已经从父类中继承了所有的属性。

对于紧密组装，由于父类与子类的属性值全部都紧密地组装在一个数据库中，将数据库中相应对象各字段的值赋予子类可能拥有的所有属性即可。这里，“可能拥有的所有属性”包括了从父类继承过程的属性，从而最终返回这一具有实际意义的对象实体。

对于连接组装，由于子类数据库中只有相对父类特殊的属性，所以除了从子类属性的保存数据库中取出子类的特殊属性值外，还要用连接找到父类对应的属性保存数据库，取出其属性值，给这一实例中的相应属性赋值。由此，即产生了一个包括全部父类、子类属性的对象。

当“取出”服务根据属性所在的类实例化出一个对象时，其空对象已经具有了该类所有的一切服务，包括从父类中继承的所有服务。当这个类所有的属性值(包括从父类中继承的)赋值完成后，一个可跨越系统的运行期而存在、系统中曾经实例化的已有的对象实体就被完整地还原出来了。组装连接可以解决多重继承的关系，一方面使各自的对象可以封装得更好，另一方面也产生了新的问题，即如何解决在组装连接中修改(无论是录入性修改还是程序触发性修改)子类中从父类继承下来的已经永久保留的属性。

3.5 所有要修改永久属性的服务均应多态

所谓多态是指一个服务有多种算法或实现，即一个相同的服务名可以具有不同的实际实现。对于永久对象属性的录入性修改，重新设计录入界面时，将属性录入并存盘即可。对于永久对象属性的程序触发性修改，修改的服务必须是虚拟的，如果属性所在的实际数据库与原服务所处理的数据库不同，必须针对实际的数据库另行设计算法，即实现虚拟。例如：有一辆“奥拓”与轿车是继承关系，系统存在2个数据库，一个是一般类轿车数据库，记录的轿车对象的已有实体，另一个是“奥拓”数据库，记录的是“奥拓”对象的已有实体，“奥拓”数据库对父类的发动状态属性采用紧密组装，系统要求对所有的轿车初始化置停止状态。在OO语言实现时，只需用一般轿车(最高层父类)的ChangeState服务遍历所有轿车(包括一般轿车与“奥拓”)即可。这样，就必须分别为一般轿车设计一个ChangeState服务，这个服务是虚拟的，修改是一般类轿车的数据库，子类“奥拓”必须针对“奥拓”的数据库结构重新编写ChangeState服务，当系统通过父类的ChangeState遍历到“奥拓”(子类)的实体时，OO语言的编译系统将根据多态的原则，认为子类的ChangeState是真正要调用的服务。

3.6 实例连接

实例连接的实现用数据库实现非常容易，实现的方法是将实例连接类建库，将实例连接的属性组成数据库结构，同时将实例连接的两端也加入到数据库结构中，使两个数据库通过实例连接数据库连接起来。如操作员与工作站之间的“使用权”实例连接，使用权数据库除了要有“使用权限”等属性外，还应加上操作员编号及工作站编号两个属性。值得注意的是，操作员编号及工作站编号并不是“使用权”类的属性，即在分析建模时，这些属性并不是“使用权”的类属性，但在“使用权”的已实例化对象实体中，必须表明“谁用什么使用权使用哪台工作站的”关系，因此记录已实例化对象实体的数据库必须有操作员编号及工作站编号两个属性。

4 结 束 语

本文提出的方法成功地解决了数据库在类的整体-部分关系、一般-特殊关系、实例连接中的应用。数据库在面向对象的分析、设计中，定位清晰、结合紧密、可操作性强，实现了数据库、面向对象中的相关概念的统一。同时，由于此方法仅利用了数据库最基本、最简单的功能，因而适应性强，可靠性很高。

参 考 文 献

1 邵维忠, 杨芙清. 面向对象的系统分析. 北京: 北京大学出版社, 2000

2 陈文宇. 面向对象的关系数据库设计. 电子科技大学学报, 2002, 31(1): 53-56

3 陈文宇. 面向对象软件的测试. 电子科技大学学报, 2001, 30(6): 613-617