Oracle8/8i 中的对象关系特性

贾代平（烟台东方电子系统所 264001）

摘要：对象数据库是目前关系数据库的一个重要发展方向，本文从应用的角度阐述了Oracle8/8i中面向对象的思想及其主要概念，并说明了如何在关系模型中运用对象特性来提高我们的应用设计水平。

关键词：Oracle 对象-关系数据库 行对象 列对象 对象视图

今天的Oracle已经不是严格意义上的关系数据库系统RDBMS，Oracle8/8i扩展了传统的关系数据库功能，使之包含了面向对象（Object-Oriented）的思想。Oracle8是具有对象关系特性的第一个版本。作为对象关系数据库系统ORDBMS，开发者的应用设计方式可以有三种不同的选择：纯关系的方式、对象-关系的方式（Object-Relational）、面向对象的方式。Oracle8/8i对这三种方式都提供了比较全面的支持，但从目前的应用情况看，纯粹的关系数据库应用占绝对的主导地位，完全采用面向对象的设计方法是不现实的，一种过渡的策略是采用对象-关系的方法来处理我们的数据库应用，本文准备从应用的角度对Oracle涵盖的对象-关系特性做一次比较全面的探讨。

1 使用对象数据库的好处

在软件开发领域，使用面向对象的设计方法已经成为开发设计的主流，那么在数据库领域，采用对象的方式与采用关系的方式相比有哪些好处呢？我看主要有如下三个方面：

(1) 对象重用：如果编写面向对象的代码，就会使得我们的代码能够方便地得到重用；同样地，如果我们创建面向对象的数据库对象，在数据库设计过程中的对象重用便成为可能。

(2) 数据表示的标准化：一旦创建了某种数据库对象，我们就可以在其他地方多次使用它们，这使得我们在不同的地方采用统一的格式来表示我们的数据。例如，如果我们创建了一个“客户”对象，那么在数据的表示上，数据库中所有的客户都将使用与此相同的内部格式。

(3) 数据处理的规范化：对象可以看作是数据及其处理方法的集合。对于每一个对象，我们可以定义在其上运行的过程和函数（统称方法），从而可以把数据和处理数据的方法结合起来。有了这种数据的访问方式，随着对象的重用，使得我们对数据的处理走向规范化。

(4) 面向对象的设计思想更符合人们处理事务的思维方式，正因为如此，在软件领域，对象的方法和理论才得到普遍的应用。

2 对象类的声明、实例的创建与初始化

面向对象的设计本质上是将对象模型直接转换为计算机应用，系统中的每一个实体在应用中表示为一个对象。对象表示了真实世界实体的属性数据以及对这些属性数据的操作。关系的方法是在一个较低的层次上使用一系列表列和表行处理数据，对象的方法是在一个更高的层次上处理数据，它处理包含着数据的对象。例如，在面向对象的数据库中处理“客户”数据时，实际上是在处理一个称之为“客户”的对象，通过对象与对象之间的关系来处理客户与其他实体数据之间的关系，而不是在处理客户及其相关联的一系列表，并且处理这些对象就象在关系数据库中处理表行与表列数据一样，能够存储、更新与检索数据。为了保持与关系数据库的向下兼容，Oracle8使用扩展的SQL作为与数据库进行通信的标准方式。为了使Oracle8支持对象特性，数据库系统在安装时需要选择安装对象选件（Objects Option），毕竟目前的Oracle其核心还是基于关系引擎的，它通过必要的扩展来支持数据库中的对象模型。在Oracle8中，实际使用的对象是通过对象

类（Class）来定义的，一个对象类描述了与一类特定对象相关的属性和方法。对象类是Oracle数据库中实现面向对象功能的起点。

2.1 对象类的定义

CREATE TYPE 对象类名 AS OBJECT (??)；

在括号中具体说明该对象所拥有的属性及其标准操作（成员函数或方法）。下面的语句定义了一个通用的地址类 Obj_Address，并说明了该地址类的几个相关属性：

```
CREATE TYPE TY_Address AS OBJECT
( Country  CHAR(3),
City VARCHAR(20),
Street VARCHAR(50),
Post_Code  NUMBER(6))；
```

对象类可以嵌套使用，下面的语句在地址类的基础上定义了一个客户类 TY_Customer：

```
CREATE TYPE TY _ Customer AS OBJECT
( Cust_Name varchar2(20),
BirthDate date,
Cust_Addr  TY_Address)；
```

说明：上述两个对象类没有定义其方法。

2.2 对象类的实例化与初始化

对象在使用前需声明一个对象类型的变量作为对象类的实例，这和在PL/SQL程序中声明其他普通变量一样，如下面的语句创建了一个TY_Address类的一个实例OBJ_Addr1：

```
DECLARE  OBJ_Addr1  TY _ Address；
```

此时的实例OBJ_Addr1实际上是一个NULL对象，还不能对其属性进行引用。在开始使用该对象前，必须对实例进行初始化，按照Oracle PL/SQL的语法规则，实例的初始化是通过一个构造函数（Constructor Function）实现的，该构造函数是在类定义时由Oracle自动创建的，且名称与类名相同，函数的参数与对象拥有的属性相对应，返回值为对象类型，上例地址类的构造函数如下：

```
FUNCTION  TY _ Address ( Country   IN   CHAR(3),
City IN    VARCHAR(25),
Street IN VARCHAR(50),
Post_Code IN NUMBER(6))
RETURN TY _ Address；
```

由此，对象 OBJ_Addr1 可以类似这样初始化：

```
OBJ_Addr1 = TY_Address('001', '烟台', '市府街45号', 264001)；
```

2.3 对象方法的定义与使用

前面举例的对象仅仅包含对象属性。一个完整的对象类应该是既有数据结构部分，也有对数据的操作部分，即方法。从本质上说，方法就是对象中包含的标准程序，对应于内部的函数或过程（成员函数）。

对象方法的定义分两部分，首先是方法命名，在类定义中说明；其次是方法的实现，通过创建类主体（Body）给出。举例说明如下：

对象类的定义部分：

```
CREATE TYPE TY_Animal AS OBJECT
( Breed VARCHAR2(25),
Name VARCHAR2(25),
Birthdate  DATE,
 MEMBER FUNCTION AGES(Birthdate IN DATE)
RETURN NUMBER)；
```

类成员主体部分，函数AGES返回以天计的年龄值：

```
CREATE TYPE BODY TY_Animal AS
 MEMBER FUNCTION AGES(Birthdate IN DATE)
RETURN NUMBER IS
BEGIN
RETURN ROUND(SYSDATE - Birthdate)；
END；
 END；
```

如果某个对象（或表）在定义时使用了对象类型TY_Animal，则该对象（或表）就可以使用其方法——AGES函数。如：

```
CREATE TABLE Animal （Id NUMBER， Animal_Info TY_Animal)；
INSERT INTO Animal VALUES (123,
 TY_Animal ('Cow', 'Mimi', TO_DATE('1998-10-23', 'YYYY-MM-DD')))；
SELECT Animal. AGES(Animal _Info.Birthdate) FROM Animal；
```

注：在插入操作时使用了对象类的构造函数TY_Animal( )。

2.4 对象方法的几种类别

方法是与对象相关的PL/SQL过程或函数，Oracle8支持如下四类方法：构造方法（Constructor Method）、成员函数（Member Function）、映射函数（Map Member Function）、排序函数（Order Member Function）。

构造方法是Oracle根据对象类的定义自动创建的一

个过程，根据这个过程我们能够在对象-关系模型中执行SQL的DML语句，如对象数据的检索、更新等。缺省情况下，对象的构造方法与其类名相同，并且将对象的所有属性作为方法的输入参数。

成员函数是与一个对象类型相关联的内嵌函数，它可以有输入参数，并且需要有一个返回值。我们通常所说的对象方法就是指这类成员函数。成员函数在定义时可以限定编译指令，以规定函数能够执行的数据操作类型。

映射（Map）和排序（Order）函数是Oracle为对象模型提供的两类特殊函数，主要是为了在对象与对象之间执行比较操作。由于对象是一种抽象数据类型，并不能看作是标量类型，传统意义上的比较操作只能是判断两个对象是否相等，象标准SQL中的比较操作如Order By、Distinct等就不能用于对象类型。为了适应对象模型中的这种操作特殊性，Oracle为开发人员提供了一种将对象类型变相转化为标量类型的函数操作，即Map和Order，这是标识对象成员函数的两个特殊关键字。Map函数和Order函数都是用于对象的比较操作，两者都返回标量类型，所不同的是Map函数不允许输入参数，返回下述标量类型之一：DATE、NUMBER、CHAR、VARCHAR2或REAL；而Order函数可以接收一个输入参数，并根据所提供的参数返回1、-1或0值。两类函数都是为了处理在不同情况下的比较操作。

2.5 类成员的管理与对象类权限

在对象模型中，对象是一种封装，它把相关数据（属性）及其对数据的操作（方法）封装成特定的对象类型，对象的内部数据主要通过定义在其上的方法来进行操作。为了对象内部数据的安全性，有必要限制对象的每一个成员函数对内部数据的操作状态，Oracle使用纯度级别对成员函数作出标记，以限制成员函数的使用范围。在对象类的定义中其限制格式如下：

PRAGMA RESTRICT_REFERENCES(方法名，约束名)

其中PRAGMA RESTRICT_REFERENCES是PL/SQL语言的编译指令，用以规定方法对数据的操作类型，即约束，它分为如下四种，且可以组合使用：

WNDS（Writes No Database State） 不写数据库状态

RNDS（Reads No Database State） 不读数据库状态

WNPS（Writes No Package State） 不写包状态

RNPS（Reads No Package State） 不读包状态

当用户创建一个新的对象类时，默认情况下该对象属于当前模式（Schema）下，如果一个用户User_1创建了对象类TY_Address，另一用户（如User_2）则没有权限使用该对象，除非显式地执行如下语句：

GRANT EXECUTE ON TY_Address TO User_2；

此时User_2就可以通过引用User_1. TY_Address来获得对类TY_Address的使用权限。

在Oracle的纯关系模型中，可以对存储过程、函数或包等授予某用户的EXECUTE权限。在Oracle的对象关系模型中，EXECUTE权限同时扩展到对象类型上。之所以在对象类型上使用EXECUTE权限，是因为在对象类（Class）上包含了方法——操作对象数据的PL/SQL函数或过程。如果授予数据库用户使用某个对象类的权限，只要对该用户授予在这个对象类上的执行（EXECUTE）权限，它同时包括了操纵对象属性和使用对象方法两个方面的权限。

3 对象模型在数据库设计中的应用

3.1 行对象与列对象

对象的概念在软件领域是普遍存在的，将面向对象的思想引入数据库系统，可以引出两个特殊的概念：行对象和列对象。

行对象就是将表的一行作为一个对象来处理，包含行对象的表称为对象表。在Oracle8中，对象表可以用“CREATE TABLE 表名 OF 对象类名”命令来建立。从这个数据定义语句可以看出，要创建对象表，事先应规划和定义好构成行对象的抽象数据类型——对象类，有了这个“类”，对象表的每一行就成为这个类的一个个实例对象，对象的属性相当于关系表的域或字段（Field）。对象表的操纵方式在很多方面类似于关系表（借助于类的构造函数），使用对象表的一个明显的好处在于事先预定义的类方法，它为设计开发人员操纵表中的数据增加了一条全新的途径。

列对象在对象-关系模型中具有普遍意义，为了扩展关系模型中字段的数据类型只能采用标准数据类型的局限性，对象-关系模型将能够自定义的对象类型延伸到对字段的定义中。如果将表的某一列定义为对象类型，这样的列就构成了所谓的列对象。列对象极大地丰富了表中的每一列对数据的表达能力，可变数组和嵌套表就是列对象的两种特殊表现形式。

3.2 对象-关系模型中的对象指针操作：REF和DEREF

REF操作符允许用户引用已经存在的行对象，它以

前面提到的对象表或行对象为操作参数，操作结果是获得指向行对象的指针，即所谓的行对象标识OID，行对象标识本身并没有给我们提供有用的信息，但通过这个值其他的对象就可以引用对象表里的行对象了。REF 有两种基本的用法：一是作为对象函数来使用，假设A为一个对象表，则REF(A)获得对象表A中的行对象标识；二是作为对象指针的标识符使用，语句“Declare X REF Y”声明了一个对象指针X，它指向行对象Y。这种用法既可以用在变量声明中，也可以用在对象的属性定义或表列的数据类型定义中。如何从这里的行对象标识OID或行对象指针获得我们需要的有用信息，就需要另外一种操作——DEREF。

REF 操作符将行对象做为它的参数并且返回其对应的引用值（OID或对象指针）。DEREF操作则完成相反的功能——它接收一个引用值并返回相应的行对象。DEREF以OID或对象指针作为其操作参数，可以使我们很容易地获得引用所指向的对象信息。

在对象-关系模型中，通过REF和DEREF操作，使我们获得如下好处：(1) 通过对象间的引用关系，可以避免关系模型中的“联合”操作。(2) 一般的多表操作，需要我们掌握表间的主外键关系，而在引用关系中，我们并不需要知道引用双方是如何“连接”的，仅仅通过OID或指针的反向操作就可获得被引用对象的详细信息。

3.3 对象模型在应用过程中的几个概念

(1) 抽象数据类型（Abstract Datatype）：与Oracle标准的数据类型NUMBER、DATE、VARCHAR2相对应，抽象数据类型是由一个或多个标准的数据类型组合而成，如果我们在其上定义了访问这类数据的方法（Methods），这样的抽象数据类型就构成了一个标准的数据库对象类（Class）。抽象数据类型可以嵌套使用。

(2) 嵌套表（Nested Table）：嵌套表是表中之表。一个嵌套表是某些行的集合，它在主表中表现为一列。对于主表的每一条记录，嵌套表中对应着多行，它是在一个表中存储一对多关系的一种特殊方法。考查一个包含部门信息的表，在任何时候每个部门都会有多个项目正在实施，在严格的关系模型中，这种一对多关系至少需要建立两个独立的表。在对象-关系模型中，采用嵌套表的方式，这里一对多的项目信息表现为部门表的某一列，这种表示方法消除了关系模型中费时的联合操作。

(3) 可变数组（Varying Array）：它是一种特殊的数据类型，用来在对象-关系模型中表示多个重复的数据项，即在一个表中可以使用可变数组作为某个字段的数据类型。可变数组的大小以及可存储数据项的性质由可变数组申明时决定。可变数组可以存储抽象数据类型的数据，因此它与嵌套表在概念上有一定的相似之处，可变数组和嵌套表是对象-关系模型中“列对象”的两种表现形式。考虑上例的项目信息，一个项目有多位职工参与，而一位职工又可以参与多个项目，在严格的关系模型中，表示这样的信息至少需要建立三个相关表（一个项目表、一个职工表、一个反映项目与职工交叉信息的表）。在对象-关系模型中，采用可变数组的方式，可以很容易地在一个项目表中记录参与项目的职工信息。

(4) 引用型对象（Referenced Objects）：可变数组和嵌套表可以看作是嵌入型对象（Embedded Objects），它们都是作为“列对象”嵌入到某个主表（或对象）中。与之相对应，引用型对象可以看作是“行对象”，与列对象不同的是，引用型对象与引用它的主表（或对象）在物理上是分离的，两者通过引用关系（指针）建立对象之间的联结。在纯对象模型中，引用型对象必不可少，它是构成对象表的元素。

(5) 对象视图（Object View）：对象视图允许开发人员在纯关系模型中增加面向对象的特性。如果我们根据已存在的关系表创建抽象数据类型（对象类），并由此构造对象视图，这样的视图就结合了关系结构和对象结构的优点。利用对象视图，我们可以在关系数据库上开发面向对象的应用，因此对象视图可以看作是关系模型和对象模型之间的桥梁，它是处理数据库应用的一种折衷解决方式。■

参考文献

1 Oracle8i: The Complete Reference The McGraw-Hill Companies. 1999
2 Oracle数据库开发指南 清华大学出版社 1998.6