基于 ODMG Java 绑定的面向对象数据库存储管理①

刘彦宇[1]　章杰鑫[2]　陈晓辉[1]

(1.桂林理工大学 信息科学与工程学院 广西 桂林 541004; 2.广西经济管理干部学院 图书馆 广西 南宁 530007)

摘　要： 针对面向对象数据库(OODB)的存储管理进行研究，依照 ODMG3.0 标准，并以 Java 绑定方式提供实现方案。通过对典型对象存储结构的研究，给出一种兼顾各种类型数据特点的分层对象存储结构。结合这里使用的体系结构，给出一种基于分槽页结构的改进文件存储结构，将存储单位由页面替换成数据库文件。分析 OODB 中引入的索引，给出一种基于 B+-Tree，结合继承层次和聚集层次的综合索引策略。

关键词： 面向对象数据库;存储管理;存储结构;文件结构;索引;ODMG Java 绑定

Storage Management in OODB Based on ODMG Java Binding

LIU Yan-Yu[1], ZHANG Jie-Xin[2], CHEN Xiao-Hui[1]

(1.College of Information Science and Engineering, Guilin University of Technology, Guilin 541004, China; 2. Guangxi Economic Management Cadre College, Nanning 530007, China)

Abstract: The paper studies the storage management for object-oriented database, and a new scheme based on the ODMG3.0 standard. is presented by Java binding. By studying the typical storage structure of persistent objects, it proposes a layered storage structure that takes into account the characteristics of diversified data type. Combining with the architecture here, it transforms a file structure based on Slotted-Page Structure, substituting storage unit from page to database file. After analyzing the introduced indexes for OODB, the paper presents a uniform indexing scheme, based on a B+-Tree and with both the inheritance and the aggregation hierarchy.

Keywords: OODB; storage management; storage structure; file structure; index; ODMG Java binding

目前对于面向对象数据存储的研究主要有两个方向。一是以关系数据库为基础，向其中加入面向对象特性，使之具有面向对象的表达能力，即对象关系数据库(ORDB)。另一个方向是以面向对象语言为基础，加入数据库特性。这种方式就是这里讨论的 OODB。主要产品有 ObjectStore、ONTOS等。主要标准是 ODMG3.0标准。包括：对象模型、对象定义语言(ODL)、对象查询语言(OQL)、C++绑定、Java 绑定、Smalltalk 绑定。本文主要针对 OODB 的存储管理进行研究，依照 ODMG3.0 标准[1]，并以 Java 绑定方式提供实现方案。

1　存储管理构架

存储管理是一个数据库系统的物理实现层，它是所有其它功能模块的基础，对于任何一个数据库系统，它设计与实现的好坏直接影响着系统的性能、系统的安全与恢复，并与整个系统的总体结构和数据模型(包括模式和持久性策略)息息相关。负责管理的数据包括目标数据、元数据、索引等。首先给出影响 OODB 存储管理设计的几个因素。

(1) 体系结构

这里提供两种模式。a)本地模式。这种模式下直

① 收稿时间:2009-11-05;收到修改稿时间:2009-12-08

接存取数据库文件，作为应用程序的嵌入式数据库，一个时刻只有一个用户、一个进程。b)客户端/服务器模式。这种模式下的常见体系结构可以分成 3 类，即对象服务器结构、页面服务器结构和文件服务器结构[2]。这里采用对象服务器结构，特点是在客户和服务器之间传输数据的单位是对象，最大优点在于可以方便地实现对象级控制，主要缺点是网络消息频繁，服务器处理开销大。这里为减轻服务器负担，在服务器端及客户端均提供对象模式信息。通过将持久类打包成 JAR 库向客户、服务器程序部署。存储时，客户端保留对象模式信息，可以在客户端将对象格式转换成同服务器存储空间中的对象格式完全一致的形式，以对象为单位直接存储到服务器文件流缓冲区，避免服务器额外的转换开销。查询时，服务器保留对象模式信息使得服务器可以执行对象查询处理，将结果集返回客户端，由客户端自行实现持久对象。

(2) 持久类描述

持久类的描述决定着存储管理的设计。Java 绑定中要求提供两种方式描述 Java 的持久类。一种是将既存的 Java 类当成持久类用。另一种通过 ODMG ODL 的前置处理器产生 Java 类描述，作为对象模式信息。Java 绑定包括了一个符合 Java 语法的 ODL 版本,对象模式信息来源于 ODL 说明。它描述了应用程序需要持久化的类型、属性类型、各类型之间关系的遍历路径、方法描述等信息。根据这个 ODL 说明生成对象模式信息。这里将对象模式信息简化表示成类元数据(Class Metadata，结构如图 1)形式，通过 Java 的持久类模型来收集。

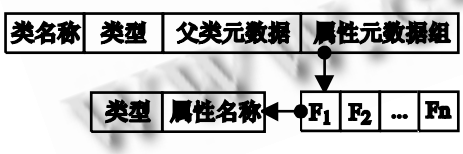

图 1 Class Metadata 结构

(3) 持久性策略

Java 绑定采用可达持久策略，当事务提交时，所有可以从根对象到达的对象都被保存在数据库中。

对象的存储由两部分组成。第一部分是对象模式信息，同一类的所有对象具有相同的对象模式，因此这些信息只需存储一次，把它存放在该类的数据字典中。对于方法实现的源代码和目标代码则分别存于系统的源代码库和目标代码库中。第二部分是该类对象的实例数据，它们被一一存放在对象库中。

这里只需存储简化的对象模式信息 Class Metadata，及对象的实例数据，对于方法描述和方法实现等信息不作为存储的一部分。因为 Java 绑定中，持久对象可以通过部署在客户或服务器程序中的持久类来实现，无需通过对象模式信息实现，所以不存储方法描述、方法实现等信息。但是，如果没有部署，即没有该对象的持久类模型，这时就可以通过简化的对象模式信息 Class Metadata 创建一个该类的模拟对象来代替，它是一个“通用对象”，用两个数组分别代表属性及这些属性的值。

最后给出这里实现的 OODB 存储管理构架，如图 2。

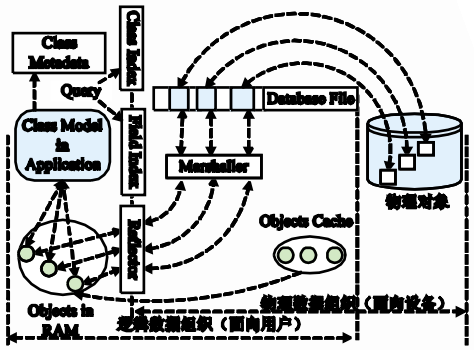

图 2 存储管理构架

●映射器(Reflector)：运用面向对象语言(OOL)的反射机制，及应用程序中的类模型来实例化对象、获取属性值、填充属性值。当内存中的对象通过 Reflector 层后，对象的属性值被获取。当数据库文件中的对象数据通过 Reflector 层后，对象被实例化，并且对象的属性值被填充。

●编组器(Marshaller)：Marshaller 利用收集到的 Class Metadata 和 Reflector 把对象转换成二进制数据存入数据库文件，或把数据库文件中的二进制数据在 Reflector 配合下实例化成对象。

●类索引(Class Index)：类索引是数据库为同一类型的所有持久化对象维护的一个以对象标识符(OID)为关键值的 B+-Tree 索引。在查询时如果没有属性索引，可以用它检索该类的所有对象。

●属性索引(Field Index)：在属性上建立的索引。

●对象缓冲(Objects Cache)：所有存储、检索的持久对象的引用。利用 OOL 中弱引用机制，把持久对

象的 OID 与引用对象的身份 ID 关联。

2 对象存储

在外存中，数据库以文件形式组织。数据库文件分配、回收的最小逻辑单位是数据块。文中出现的 Address 是相对地址，即数据块的地址。数据在数据库文件中的绝对地址为：Address × 数据块长度(BlockSize)+地址偏移量(AddressOffset)。这里利用 OOL 中流方式向磁盘文件写入、读取字节。文件流内部已具有自动缓冲，也可以设置缓冲大小，不用通过增加 BlockSize 来控制磁盘 IO 数，所以将 BlockSize 置为 1，既不会浪费空间，又不会增加磁盘 IO 次数。这里 BlockSize 的设置主要用来增大数据库文件的最大容量，为 1 字节时数据库文件最大可增至 4GB，为 8 字节时可增至 32GB。另外，文中出现的 Size 为数据块个数，实际占字节数：Size×BlockSize。对象存储包括三方面内容。

2.1 对象数据的存储结构

在 OODB 中，一个数据库文件可以存储多种不同类型的对象。一个对象的组成成分，即属性，可以是一个简单类型、集合、结构、甚至是一个对象，允许嵌套集合、嵌套结构、嵌套对象，对象的嵌套层次不限。因此对象属性的长度，对象的长度均是变长的。另外，OODB 中对象的存储也不能简单的沿用传统的数据库技术。对于合成对象，采用把主体对象与成员对象分开存储的策略。

典型的对象存储格式是由属性个数、具有明确指定值的对象的所有属性标识符(V1 V2…Vn)、属性在对象存储格式的 Values 部分的偏移量(O1 O2…On)、Values 值等部分组成[3]。在实际应用中，(Vi Oi Value)这种格式对不同类型数据采用相同存储格式，很不灵活，没有兼顾各种类型数据特点。

对于继承对象则常采取分段存储策略[4]。用抽象格来描述父类、子类之间的语义联系。对于一个抽象格中的对象，其信息项的存储是按抽象类分段存放的。每个段由直接定义在该抽象类中的那些信息项(不包括从其父类继承来的信息项)组成。这些分段分别存放在相应抽象类所对应的数据文件中。它的优点：可以快速从某个类中找出以它为根的类层次上满足条件的所有对象、能很好的支持对象变迁、方便大对象处理。但采取分段存储策略，增加了存储对象的开销，需要将对象拆开，存入不同的抽象类所对应文件中。同时也降低了对象查询效率，尤其是要访问该对象所有抽象类的分段数据。在实际应用中，这种分段存储策略增加了磁盘 IO 次数，效果并不好。

这里给出一种策略，保持继承对象的整体性，但是采取分层存储，使对象结构清晰。在支持继承层次索引、对象变迁方面，将继承对象 OID 通过 Class Index 方式加入它的各层抽象类中，这样每层抽象类都是一个包含子对象的超集。

下面给出这里实现的对象存储结构。

首先，给出 ODMG3.0 对象模型支持的类型，包括对象类型、文字类型[1]。对于不同类型数据采用不同的存储格式。

(1)原子类型：包括没有定义内部的原子对象类型。对原子文字类型则支持整型、浮点型、字符型、布尔型、枚举型、字符串型等类型。对于除字符串型外的其它原子类型，直接存储。对于字符串型则采用(基地址(BA),负载长度(PL),值(Value))格式存储。BA 是属性在整个对象存储格式中的偏移量，PL 是 Value 的长度，Value 包括字符元素个数、字符串值两部分。

(2)聚集类型：包括集合类型、包类型、列表类型、数组类型、字典类型。对于聚集类型采用(BA,PL,Value)方式存储。其中 Value 包括聚集类型种类 ID、元素个数、各元素值。如果元素为除字符串外的原子类型，则元素值直接存值。如果元素为字符串或聚集类型，则元素值存储该元素 BA，PL。

(3)结构类型：日期类型、时间间隔类型、时间类型、时间戳类型、Struct 自定义的结构文字类型。在 Java 绑定中结构类型映射成 Java 类。对于类实例，存储其 OID。

最后，通过存储一个 OOdb 类(如图 3)对象来阐述这里实现的对象存储格式，如图 4。OOdb 对象既是一个合成对象，又是一个继承对象。其中对象属性包括 ODMG 对象模型支持的各种类型，像原子类型整型、字符串型，聚集类型数组，以及对象。

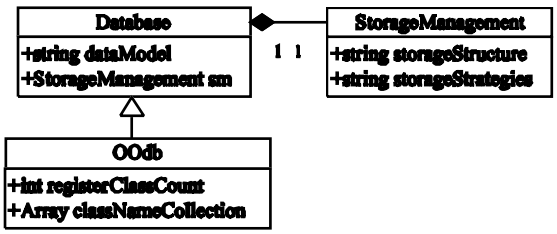

图 3　OOdb 类

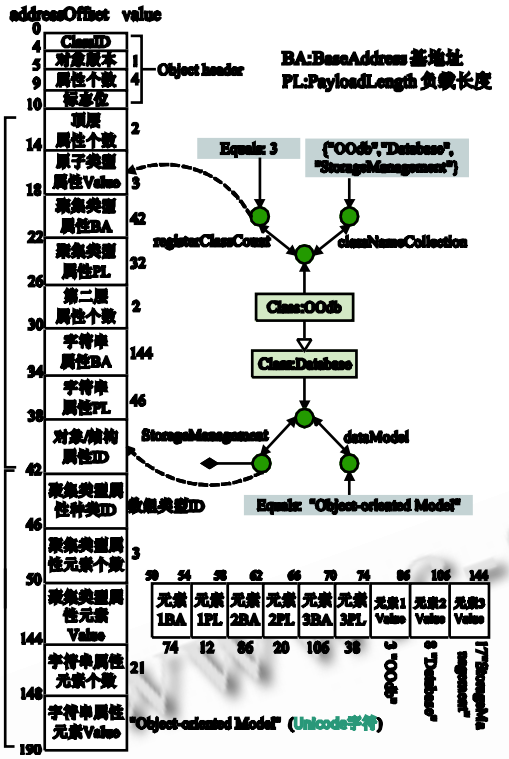

图 4　OOdb 对象存储格式

2.2 对象数据的存储组织

在关系数据库中，一个文件往往包含成千上万条记录，这些记录在文件上如何存放，如何安排，这是文件中记录的存储组织问题。常见文件结构包括顺序文件、散列文件等。

把记录放在页面中存储组织时，常采用一种能表达变长记录的页面格式，分槽页结构[5]。在页面中存放记录的基本思想是在页面的首部页面头之后按从前往后的方向存放(记录偏移，记录长度)二元组，每一个二元组我们称之为记录槽(Slot)，而在页面尾部按从后往前的方向存储记录本身。插入总是从空闲空间尾部开始。删除记录时，只需把页首部的该记录大小改为-1 即可，然后把其左边记录移过来填补，将页面进行紧缩操作。

这里每个对象数据可看成变长记录，所以必须采用能表达变长记录的组织方式。同时，由于采用对象服务器结构，客户和服务器之间数据的传输单位是对象，使我们可以直接将对象数据传给服务器，不需要使用页面来组织对象，而服务器又通过文件流缓冲区存取数据，可看作直接操作数据库文件。基于上述原因需要把这种分槽页结构应用到数据库文件上，即把存储单位由页面替换成数据库文件。替换后需要解决一个问题：空闲空间的管理。在分槽页结构中删除记录后，将页面进行紧缩操作，由于页面不大，所以移动代价不高，而数据库文件则很大，不适合这种操作。下面给出改进后的空闲空间管理算法。

(1)空闲空间管理：数据库运行期间对象的不断存储和删除，会出现许多“碎片”，如何管理这些空闲槽使空间利用率高，分配速度块。这里并不调整对象数据位置来填充，考虑到空间分配效率，为分散在数据库文件中的空闲槽在内存中建立 AVL 搜索树，实现满足大小要求的最小空闲槽的快速检索。如果数据库文件没有空闲空间，那么对象数据被附加写入文件的末尾。

(2)关联 AVL 搜索树，如图 5。系统为所有的空闲槽建立两个 AVL 搜索树。空闲槽地址树(FreeByAddress)以空闲槽 Address 作关键值进行排序，空闲槽大小树(FreeBySize)以空闲槽 Size 作关键值进行排序。为这两个树中的每个节点都增加一个 Peer 指针，指向该空闲槽节点在另一个树中的对应节点。当分配空间时搜索 FreeBySize 树，然后根据找到节点的 Peer 指针检索相应空闲槽 Address。如果找到的空闲槽 Size 大于要申请的空间 Size,则将剩余空间形成新的空闲槽，插入搜索树。当释放空间后空闲槽 Address、Size 分别加入两个树中，并关联，然后根据 FreeByAddress 树判断能否动态合并。如图 5，当 OOdb 对象被删除时，空间被释放，形成空闲槽，将该槽 Address 加入 FreeByAddress 树，并检测该结点的前后结点，通过 Peer 查找前后结点 Size，判断能否合并。本例中存储 OOdb 对象的 Size 为 190，释放后该空闲槽刚好可以和前后槽合并。

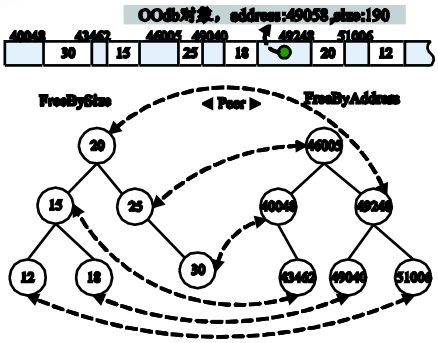

图 5　关联 AVL 搜索树

在查找方面，Slot 中存储(对象 Address,对象长度)而不是(记录偏移，记录长度)。使 OID 等于 Slot 地址，可以直接读写任意一个对象，便于文件的增、

删、改。下面给出 OID 实现策略。

(3)OID 唯一标识数据库中的每个对象。分为物理 OID、逻辑 OID、混合式 OID。其中逻辑 OID 的值与对象的物理地址之间存在某种映射关系，这种方式灵活，便于移动对象。这里采用逻辑 OID 的策略，使 OID 直接与数据库文件中的地址指针相等，地址指针指向 Slot。这样确保任何对象只需两次读数据库文件操作就可以被检索到，如图 6。这种方式无需维护映射表。

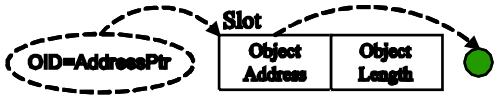

图 6　通过 OID 访问对象数据

这种基于分槽页结构的改进文件存储结构插入简单，支持变长对象的存储，支持空闲空间动态合并，Slot 地址与 OID 的绑定又加快了文件中对象的检索速度。

2.3 数据字典的存储

数据字典用来存储数据库中对象数据的描述信息和管理所需的控制信息。这里实现的数据字典存储结构如图 7。

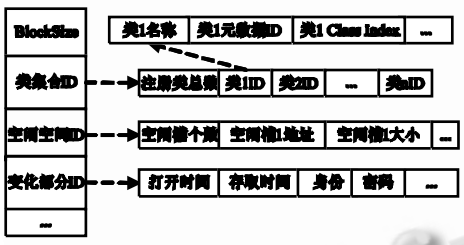

图 7　数据字典存储结构

3　索引结构

在 OODB 中，属性索引的实现需要重点考虑两方面的内容。

(1)继承层次索引：OODB 中，一个类可以特殊化成一系列递归子类，形成一个继承层次。因此，查询目标既可以限定在单个类上，也可以扩展到以该类为根的继承层次上。结果导致 OODB 索引不仅要支持单个类上的对象检索，还要支持这个继承层次上所有对象的检索。支持继承层次索引的技术包括，类层次索引(CH-Tree)，H 树，HcC 树，和 CG 树，类分组(CD)策略，基于数据库统计数据的方法，二维类层次索引等。但是到目前为止仍然没有一种策略可以和满外延(full extent 以下简称 FE)索引配置[6]的检索性能相当。一个类的 FE 是该类及其全部子类的对象集合。FE 索引配置为每个类的 FE 建立 B+-Tree 索引。所以它是继承层次上的最优检索方案。这里采用这种索引配置。但是，这种索引配置增加了存储开销，更新开销，像添加(或删除)一个叶子类的对象，需要更新继承层次上全部祖先类索引，降低了 OODBMS 的性能。

(2)聚集层次索引：OODB 中，针对一个类的查询可以包括针对该类嵌套属性的查询谓词，这些谓词称为嵌套谓词。处理嵌套谓词的有效评估，大多数方法都是基于连接索引技术。常用策略有嵌套索引、嵌套继承索引、多重索引、继承多重索引、路径索引、路径索引的变体等[7]。其中多重索引为聚集层次中的每个类都维护一个索引，当查询时，遍历这些简单连接序列。这里采用多重索引策略，对于多重索引的维护就是对每个 B+-Tree 索引的维护，维护开销很低，但是检索性能不及单个的嵌套索引。

这里为每个类维护一个 OID 的 B+-Tree 索引，即前面提到的 Class Index，用来检索这个类的全部对象，实际上通过 Class Index 查询对象相当于无索引查询。同时在对象属性上建立单一属性索引，同样选择实现 B+-Tree，将属性值与对象的 OID 关联，对于继承属性采用 FE 索引配置，对于嵌套属性采用路径长度为 1 的多重索引。

最后将 Class Index 与 Field Index 的检索性能作对比。数据库初始包含 100 个 OOdb 对象(如图 3)，在继承属性 dataModel、嵌套属性 storageStructure 分别设置索引。如图 8 所示继承属性的检索时间很平稳，在 0-15 ms 之间，嵌套属性检索性能略好于 Class Index。

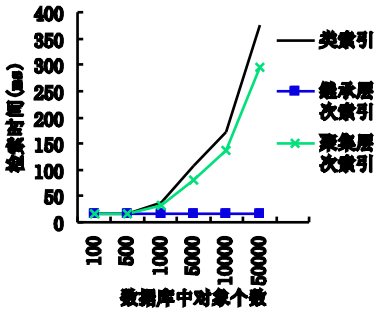

图 8　Class Index 与 Field Index 检索时间对比

4 结语

数据模型直接影响数据库的存储管理，因而OODB 的存储管理不同于关系数据库。本文通过对OODB 中存储管理的研究与实现，得出一些具有实用性的方案。但对某些提高存取效率的策略，如聚簇、高速缓冲方案等还有待进一步研究与应用。对一些依赖 OODB 存储管理的技术性问题，如对象模式演化，XML 数据管理等方面也有待进一步探讨。

参考文献

1 Cattell RGG, Barry D K. ODMG3.0, The Ob ject Data Standard.

2 于戈,王国仁，金泰勇，牧之内显文.一个 NT 平台上分布式对象数据库系统.软件学报, 2002,13(4):719－725.

3 郑刚,唐红梅.面向对象数据库中数据模型及存储结构的研究.计算机工程, 2002,13(4):719－725.

4 刘丽. 面向对象数据库模型、存储及查询优化的研究[硕士学位论文].青岛:山东科技大学, 2004.

5 Ramakrishnan R, Gehrke J. Database management Systems. 2nd ed., New York: McGraw-Hill, 2000:218－224.

6 KIM W. Introduction to object-oriented database. Cambridge, MA: MIT Press, 1990:89－125.

7 Huang YF, Chen JM. The study of indexing techniques on object oriented database. Information Sciences, 2000,130(1):109－131.