计算机科学 1996 Vol. 23 №. 2

如何在OODB中支持完全的C++类型系统*)

陶 伟 麦中凡

(北京航空航天大学计算机科学与工程系 北京 100083)

摘 要 Full support the compelete C++ type system includes virtual functions, inheritance, polymorphism, encapsulation, parameterized types (template), pointer reference. This paper presents problems of supporting compelete C++ features, and our seamless integration solution and experience.

关键词 C++, Object-oriented database, Type system, Seamlessness.

一、引 言

尽管C++的OO机制(如继承、多态、虚函数)提供了强有力的数据抽象机制,对象之间的指针引用表达了客观世界客体之间复杂的联系,成为大多数ODBMS必须支持的一种语言。然而C++毕竟是一种程序设计语言,而不是数据库操作语言,在运行时刻不能提供模式(类)信息,其诸多特征,如虚函数、继承、多态等,都依赖于特定的编译器实现。编译器为了实现C++的OO机制,往往向对象中加入一些附加信息,如虚函数表指针,造成了对象逻辑上和实际的物理内存映象之间差异,成为ODBMS实现过程中的主要障碍。

大多数的ODBMS,如Versant[1]、ONTOS[2]、O2[3]、ODE[4]、ObjectStore[5],对C++类型系统的支持是不完全的,不是无缝的。通常的做法有三种:第一,对C++做某种限制,如不能用模板(ObjectStore只能对系统提供的集类使用),不准或不支持在属性中出现指针引用;第二,让用户显式地提供某些相关的代码;第三,对C++做少量的扩充,完全容纳C++的类型系统,由预处理器完成相应的处理。方案一丧失了C++的某些特征,方案二增加了用户的开发负担,方案三使扩充的C++程序调试成为难点。我们采用的是方案三。

本文的背景是我们正在开发的多媒体智能数据库系统(MIDS/BUAA)[6],其核心是一个基于client/server体系结构的面向对象数据库。它的数据库操作语言是P++,P++[7]是基于C++的数据库编程语言,是C++语言的一个超集。P++语言提供了对象模式定义、持久化、迭代、查询、版本、约束、触发器、规则表示等重要的语言功能。P++程序由P++预编译器pfront翻译成纯的C++程序语言,这些C++程序通过调用系统的对象管理库来实现P++的语义,然后通过C++编译器进行编译和链接,以形成最终的可执行代码。pfront的另一个重要功能是提取数据库模式。

二、主要问题

1. 对象隐含指针

C++编译器在编译过程中会向C++对象中加入所谓的隐含指针,用以实现C++的继承机制,如虚函数(virtual function),虚基类(virtual base class)。C++指向父类对象的指针可以指向派生类对象,因此虚函数的调用在运行时刻才能确定。C++为了实现上述指针的多态性,为每个类生成了一个虚函数表vtbls,用于存储各个虚函数的地址。当一个类的构造函数被调用,C++编译器产生的代码会自动把虚函数指针指向vtbls入口,把虚基类指针(vbase pointer)指向相应的虚基类对象。由于隐含指针是指向内存空间指针,仅在程序运行时刻有效。当对象存入数据库中隐含指针就丧失了原有的语义。有效地保证隐含指针语义是ODBMS所要解决的重要问题。

2. 对象之间的显式的指针引用

*)航空科学基金和国防科工委预研项目基金资助。陶伟 博士生,主要从事多媒体智能数据库和面向对象程序设计等方面研究。麦中凡 教授,主要从事程序设计语言、面向对象方法学、软件工程、数据库等方面研究。

在C++程序设计语言中，指针实际是数据对象的内存的物理地址(或虚存地址)，在一般的情况下，ODBMS无法直接存储对象指针，而是通过逻辑地址来表达对象间的指针引用语义。指针的追踪，会导致内外存对象的数据交换，数据格式的转换，包括外存逻辑指针和内存物理指针之间的转换。这部分工作是ODBMS的核心功能，直接影响ODBMS的空间效率和时间效率。

3. 模式的提取

C++的类型检查是在编译时刻实施的，在运行时刻程序无法获得对于ODBMS的对象查询和操作都是必要的类的信息。

4. 对象的内存映象

C++与C不同。C语言中一个struct中的数据成员与其内存映象是一致的，而在C++中由于引入了继承机制，基类的数据成员在内存的分布并不一定按照逻辑上的次序，而随着编译器的不同而不同。隐含指针的引入亦是造成对象逻辑布局和内存布局之间差异的重要原因。对象的内存布局在编译时刻一般是得不到的，只有在运行时刻才能获得。

从上述的四个方面的分析，我们可以看出问题的根源是C++对象和数据库对象分别处于不同的地址空间，有着不同的数据格式。问题的核心是指针的语义维护。隐含指针代表着对象的操作语义，显式指针代表着对象间复杂关系。

三、解决方案和关键技术

由问题的根源出发，我们认为解决问题的关键是如何为数据库对象和C++对象提供一个统一的存储空间，同时保证数据库对象的显式指针和隐含指针的合法性。

我们的基本思想是：利用操作系统的虚存管理机制，把数据库映射到虚存空间，使数据库对象和C++对象共处于统一的地址空间，同时动态地重定位对象显式指针和隐含指针，保证数据库对象和C++对象的语义一致性。实现该方案的两大关键技术是：

1. 基于虚存映射VMM的存储技术

大多数UNIX操作系统，如SVR4、OSF/1、Berkeley BSD 4.3、SUNOS等，均提供了由硬件实现的虚拟存储系统，操作系统以系统调用的方式把这一机制提供给软件开发人员。虚拟存储系统提供了一种统一的虚存数据的地址引用，这样无论数据在主存，还是在辅存之中，其表示形式均一样，无需内外存数据格式的转换。利用虚拟存储系统，我们可以直接把数据库映射到机器的虚存空间。

采用上述方法的技术核心是采用UNIX系统提供了一整套的存储管理机制，这些系统调用赋予了应用程序对地址空间的完全控制，还允许对系统进程地址空间和各种存储对象进行各种操作。这一套存储管理机制主要由一组系统调用构成[8]，这些系统调用的函数原型均在mman.h中，其中最重要的系统调用是mmap，其功能是映射存储页面。综合地使用这些系统调用，便可以建立一种按虚存地址直接访问数据库对象的方法。

2. 通过截取SIGSEGV信号处理Page fault(或Object fault)

UNIX的虚拟存储系统允许我们对虚存地址空间的页面进行保护，包括no access，read only，execute，和read/write。当应用程序访问一个no access页面或向一个read only页面写，会导致一个Page fault，操作系统会发出SIGSEGV信号，SIGSEGV信号的处理程序可以由用户定义。页面保护和SIGSEGV信号处理的综合应用，便可以顺利完成Page fault(或Object fault)的处理。处理过程大致是这样：在Client端数据库的相应部分，例如：一个段，被映射到虚存空间，这部分虚存空间设为no access，一旦访问这些页面的对象，操作系统会发出SIGSEGV信号，用户定义的SIGSEGV信号处理程序负责从server取回所缺的页，并设为read only。如果向一个read only写，同样SIGSEGV信号处理程序会被调用，设该页为*read/write*，同时设置修改标志。这样，我们就顺利完成Page fault的处理工作。

实现该方案的重要细节如下：

1. 隐含指针的处理。隐含指针的处理在SIGSEGV信号处理程序里完成。当一个Page fault发生，SIGSEGV信号处理程序会发现并从server取回所缺的页，然后根据对象布局图，依次将该页中对象的隐含指针重定位。重定位程序是由pfront产生，并链入应用程序之中。基本处理过程如下：

(1)我们定义一个特殊的class，重载new，其目的是调用类的构造函数，但不分配空间。

```
class NOTHING{};
void * operator new(size-t size,NOTHING *p)
{
  return(void *)p;
}
```

(2)我们需要定义一个全程变量Is-Fix-Hidden-Pointer和一个宏。

```
int Is-Fix-HiddenPointer=0;
```

```
#define FIX-NOW\
    if(Is-Fix-HiddenPointer)\
    return;\
```

(3)重定位程序的基本框架如下：

```
fix-hidden-pointer(int class-type,void * p)
{
   Is-Fix-HidderPointer=1;
   switch(class-type){
      case 0:new(NOTHING *p)class-0;
             break;
      case 1:new(NOTHING *p)class-1;
             break;
      ......
      case n:new(NOTHING *p)class-n;
             break;
      }
   Is-Fix-HiddenPointer=0;
}
```

(4)pfront 把宏 FIX-NOW 放入每一个类的构造函数的首行，如果类没有构造函数，则缺省地生成一个。FIX-NOW 的目的是当重定位程序调用构造函数时，不改变对象的原有属性。pfront 生成的代码是用户看不见的，因为 pfront 实现了源代码级的调试。

2. 对象内存映象的提取。对于某类 Class-A 的对象内存映象，我们最关心的是其父类数据成员对 this 的偏移量，Class-A 中各个对象对 this 的偏移量，Class-A 中各个对象指针对 this 的偏移量。由于对象的内存布局只有在运行时刻才可以获得，我们开发了一个工具 p-map 来自动生成一段代码，经编译运行，通过对创建实例的地址运算来获得对象内存映象，内存映象的数据存入一个头文件中，链入应用程序中。p-map 要把用户程序拷贝到一个临时文件中，把所有的 private 属性改成 public 属性。p-map 由 pfront 来调用。对于模板类，不同的类型参数产生的新类其内存映象是不同的，要作为新类来处理。

描述对象内存映象的主要数据结构如下：

```
struct A-E-P {int offset;//指针的偏移量
   short int count;//指针数组的个数
   };
struct A-O-F { int offset;
                //子对象或父类成员的偏移量
    int class-id;//子对象或父类成员的模式编号
    short int count;//数组的下标
    };
struct CLASS-MEM-MAP {//对象的内存布局
   int class-id;//模式编号
   int A-E-P-n;//指针的个数
   A-E-P *explict-pointer-tab;
   int A-O-F-n;//子对象和父类成员的个数
   A-O-F *object-tab;
};
```

例如：

```
class A {…};class B:public A{…};
class C:public B{ int *p};
class C C-obj;
```

则 class C 对象的父类 class B 的成员的偏移量为：

```
(unsigned long)((B *)&C-obj)-(unsigned long)
   (&C-obj);
```

则 class C 对象的 p 的偏移量为：

```
(unsigned long)(&(C-obj. p))-(unsigned long)
   (&C-obj);
```

3. 动态地址重定位。一般情况下机器虚拟空间可以开得很大，例如 SUN3的一个进程可以开256MB 大小的虚拟空间，但毕竟是有限的，不可能囊括所有的数据库，这是其一；其二是，两个独立的数据库可能在创建时使用的是同一块虚拟空间地址，如果一个应用程序同时打开这两个数据库则会发生地址冲突。为了解决这两个问题，我们必须采用一种动态的虚存映射方法，我们需要一个虚拟地址映射表 VAM(Virtual Address Mapping)，VAM 记录了数据的相应的段对应的是哪一块虚拟地址。随着应用程序访问更多的数据库对象，新的虚拟地址空间将会被分配，直到事务结束。事务结束后，VAM 被重置，开始下一个事务的动态映射。

4. 显式指针的 swizzling 和 unswizzling。由于采用动态地址重定位，数据库的某一段映射到虚拟地址空间，其地址可能和创建该段的虚拟地址不同。例如段 A，创建于0x10000处，其中某页的指针 p 为0x10100，当段 A 第二次被访问时，映射到0x10010处，则 p 的值加10，才能成为有效指针。这就是所谓的 swizzling。当 p 所在的页某些对象的属性被修改，则该页要被写回数据库，写回之前，该页所有的显式指针均减10，这就是所谓的 unswizzling。

swizzling 操作是在 SIGSEGV 信号处理程序里完成，unswizzling 操作在一个事务结束时对所有写回数据库的页面实施。在实施指针 swizzling 和 unswizzling 操作时，需要两种信息，一是对象布局图，它提供每个页面的对象分布情况，二是对象内存布局，它提供了指针在对象中的偏移量。操作粒度以对象为单位，基本算法如下：

```
swizzling(void *p,int class-id,long delta)
//p 是对象的地址,class-id 是对象的类型,delta 是
   虚存地址差值
{
   由 class-id 获得对象内存映象;
   所有的显式指针加 delta;
   由 class-id 获得所有成员对象和父类成员的地
   址和 class-id
   递归调用 swizzling
}
```

对于 unswizzling，只需把 delta 加一负号即可。

5. Client 的页面缓冲区。它采用一个特殊的设备文件/dev/zero，/dev/zero 给调用程序一个零缓冲的虚存块，其大小由 mmap 系统调用来说明。/dev/zero 是一个特殊的设备，它对 read 系统调用的响应

是一个具有零值的无限字节源。当/dev/zero 被映射后，它建立一个未命名的对象返回给内存中的映射区域。

6. 源代码调试。P++的程序可以直接在系统提供的 debugger 中进行源代码调试，这主要是通过宏 #line 来实现的。

四、结论

ODBMS 支持 C++完全的类型系统，其核心是对象中各种指针的语义的维护。指针的语义的维护主要是通过 swizzling 和 unswizzling 操作完成的。充分利用操作系统的虚拟存储管理，可以极大地提高系统的效率，在相当的程度上减少了内外存数据对象之间的差异。

参考文献

[1] Product Profile, Versant Object Technology Co., 1990

[2]ONTOS DB 2.2 First Time User's Guide, ONTOS, INC. Feb. 1992

[3]R. Agrawal et al., Ode (Object Database and Environment): The Language and the Data Model, Proc. ACM SIGMOD 1989 Int'l Conf. Management of Data, Porland, Oregon, May-June 1989

[4]O. Deux et al., The O2 System, CACM Vol. 34, No. 10. 1991

[5]C. Lamb, et al., The ObjectStore Database System, Same to [4]

[6]Dunren Che, Zhongfan Mai, The Overall Design of MIDS/BUAA: a Multimedia Intelligent Database System, ICYCS'93, July 1993

[7]车教仁，麦中凡，MIDS/BUAA 语言的设计与实现。第十一届全国数据库学术会议论文集，1993.9，西安

[8]UNIX 系统 V 第4版程序员指南：系统服务和应用软件打包工具。电子工业出版社，1992年11月

（上接第71页）

2. 多媒体数据库的查询

查询处理是一个 DBMS 必备的特征。因此，多媒体 DBMS 同样也要能够处理查询。但是，超文本中的浏览不是严格意义上的查询。一个 DBMS 的查询包括查询条件、访问路径（显式或隐式）、结果模式。因而，处理一个查询须有条件识别、数据获取（含优化）、输出结果三个步骤。

下面，我们着重讨论查询条件和条件识别。多媒体数据库的查询条件可分为五类：①结构化数据的简单逻辑条件。是算术、字符的逻辑表达式，是传统 DBMS 所处理的查询条件。如：职称＝"教授".AND.工资＜600。②结构化数据的项匹配条件，用于 DATALOG 查询，如：:-?ancestor(李明,X)。③结构化数据的合一条件查询。用于带函数的 DATALOG 查询。如：:-?ancestor(name(李,_),X)。④非结构化数据的模式匹配条件，是由多媒体数据引起的，如例1中列举的查询。⑤模糊条件，出现在高智能化的系统中，如："找出所有的胖人"。

结构化数据的条件容易识别，最复杂的合一条件由 AI 领域中的 A＊算法处理，也容易实现。然而，模式匹配条件目前还不能很好处理，即使是采用神经网技术，模式识别的准确率也还不高，DBMS 处理模式匹配条件的查询，恐怕还有待于模式识别领域取得更大的进展。至于模糊条件识别，可望在模糊 DBMS 研究领域取得进展。

对于多媒体 DBMS，上述五种条件的查询都是可能的，并且模式匹配条件的查询将是大量的（如例1）。除此之外，多媒体数据库查询还具有递归性。为减少存储数据的冗余和数据版本的管理，一个多媒体数据对象往往分解成若干较小的对象，而小对象又可分成更小的对象，…。这样，当查询一个对象时，可能会递归地引起许多对象的查询[10]。递归查询处理，我们已经有了 DATALOG 的经验。但我们也知道，对于带函数的 DATALOG 程序，不动点的收敛性都不能保证，何况是多媒体数据对象。多媒体数据库的递归查询的收敛性是难以处理的。另外，文[11]中也证明了递归查询的固有低效性，这对于多媒体数据库的递归查询效率就更低了。虽然随着 GB 级的主存的使用，当前的许多 DB 可以全部装入主存，大大提高一般递归查询的效率，但是将多媒体数据库全部装入主存还需较长时间的努力。（参考文献共11篇略）