青鸟Ⅱ型系统面向对象语言CASE C++的设计和实现*

邵维忠 袁曙涛 杨芙清

(北京大学计算机科学与技术系 北京 100871)

摘要 CASE C++语言是面向对象的CASE环境青鸟Ⅱ型系统中设计、实现的一个与C++完全兼容并支持永久对象的面向对象程序设计语言.在青鸟Ⅱ型系统中,它是对象管理系统OMS(object management system)的操纵语言和所有集成工具的编程语言,同时也是支持最终用户的通用的编程语言.本文首先简要介绍了CASE C++的设计背景.然后,较详细地讨论了CASE C++中新引入的语言成份:永久对象;类定义的共享机制;永久对象间的关系描述——链和对象的内容等.最后,给出了实现方法.

关键词 面向对象,永久对象,C++语言,CASE环境,对象管理系统.

自从Smalltalk－80[1]诞生以来,面向对象程序设计语言以其支持封装、继承等一系列显著的面向对象思想迅速赢得了用户的青睐.面向对象程序设计语言不仅对软件生产率的提高,而且对软件质量的保证都提供了良好的语言支持.在众多的面向对象程序设计语言中,C++[2]以其支持面向对象技术和"更好的C"这两个鲜明的特色,在世界范围内获得极大的成功.但是C++也有一些不足之处,比较突出的一点就是C++不支持永久对象的定义与存储,从而给用户在保存对象和共享对象上带来了困难.

CASE C++是与C++完全兼容的、支持永久对象的面向对象程序设计语言.研制CASE C++的动机源于对面向对象CASE环境青鸟Ⅱ型(以下简程JB2)[3]中的数据集成的研究.在JB2中,工具集成要求工具之间共享数据且交换信息,同时需要管理不同工具所产生的数据对象之间的关系.为此,在JB2中设计和实现了对象管理系统(OMS).[4]OMS为工具的开发者和最终用户提供了统一的面向对象技术支持,负责永久对象的管理和存储.CASE C++语言是由OMS支持的永久性面向对象编程语言,也是OMS的操纵语言.用户在CASE C++中可以直接声明永久对象,以及它们之间的关系.所有关于永久对象的读写、并发控制等都对用户透明.CASE C++在支持面向对象技术上对C++主要有以下一些扩充:

---

* 本项目得到国家"八五"重点科技攻关计划的支持.作者邵维忠,1946年生,教授,主要研究领域为软件工程,面象对象方法.袁曙涛,1962年生,助研,主要研究领域为面象对象技术,软件工程,程序设计语言.杨芙清,女,1932年生,中国科学院院士,教授,主要研究领域为操作系统,软件工程,软件工程环境,面向对象方法等.

本文通讯联系人:邵维忠,北京 100871,北京大学计算机科学与技术系

本文1994－12－06收到,1995－02－23定稿

• 永久对象

CASE C++支持永久对象的定义、使用和存储.

• 永久对象间的关系描述——链

采用CASE C++语言提供的链机制不仅可以描述永久对象间的可达关系,而且可以描述其关系的语义.

• 对象的内容

CASE C++中引入对象的内容机制以描述对象的可变长属性,从而增强对象的表达能力.

• 类定义共享

CASE C++在类库的支持下允许不同用户或程序之间共享类的定义.

• 对象存取控制

CASE C++提供对象存取控制机制.

本文对CASE C++中这些机制作一详细的讨论,最后给出我们的实现思想.

1 永久对象

对永久性面向对象程序设计语言(POOPL)的研究是当前面向对象程序设计(OOPL)研究的一个热点.目前研究POOPL的方法主要有3种:第1种是在原有OOPL的基础之上引进一个新的永久性基类,然后通过继承而派生出永久对象.这样永久性不是语言的一个有机成份,因此每一次应用,处理它的方式都有所不同.这类方法常常通过一个类库来实现,如PC++.[5]第2种是把OOPL中的所有对象都看成是永久的,如在SMALLTALK的基础上建立OODB[6]等.第3种是在面向对象语言中增加永久性声明句柄.用户可以根据自己的需求,把任一对象声明为永久的,即是PS-ALGOL[7]中提出的所谓正交永久性的概念.

以上3种方法中第1种方法实现简单,但用户(程序员)使用起来不方便.第2种方法实现起来存储开销大,对象的存储管理复杂.第3种方法事实上是前2种方法的一种折衷,以增加语言成份来换取存储管理的简明,即系统只需管理那些具有永久存储特性的对象.CASE C++的永久对象的概念就是在此方法的基础上加上我们自己系统的一些特点而实现的.

1.1 永久对象的标识

尽管永久对象的标识(POID)的具体实现技术与语言本身无关,但不同的实现技术都将对语言中如何表示永久对象有所影响.在已有的系统中主要有以下3种标识技术:核心转储、显示标识、可达性模型.

核心转储是目前最简单的方法,它允许所有对象是永久的.当一个程序终止时,正在被使用的所有有效对象被当作一个单独的存储块保存起来.当需要访问对象时,则恢复整个被存储的块.

显示标识是通过显示的句柄说明对象是永久的,如Amber.[8]

可达性模型是指从某一指定的永久对象出发直接或间接可达的对象均为永久的.所指定的永久对象被称为"根"对象,如O2.[9]

在CASE C++中我们采用了显示标识和可达性模型相结合的方法.

1.2 CASE C++中的类

在 CASE C++中引入新的类声明保留字 CLASS,CLASS 与 C++的 class 并存,也可以代替 class 使用,但在下列 3 种情况下,必须使用 CLASS:(1)包含了链和内容的类;(2)用来定义永久对象的类;(3)要移出(export)的类.

1.3 永久对象模型

在 POOPL 中,永久对象和普通对象是采取不同的对象模型还是采用统一的对象模型是一个引起争论的问题.采取统一的对象模型的好处是支持正交永久性,并带来在程序设计层次上的对象的统一.但是系统难以支持其它语言.而对永久对象采取独立的对象模型可以使系统独立于语言,带来的好处是适应面广,可支持多种语言的实现.

在 CASE C++中,永久对象模型在以下几点不同于临时对象模型:

- 增加了对象间的关系描述——LINK 声明
- 增加了对象的内容属性——CONTENT 声明
- 引入永久对象声明符 persistent
- 不允许有静态变量和指针变量

下面是永久对象模型的一般描述:

```
CLASS〈对象类名〉{
        〈成员表〉;
        {〈LINK 声明〉}可选;
        {〈CONTENT 声明〉}可选;
        };
persistent〈对象类名〉〈永久对象名〉;
```

在永久对象模型中不允许有指针变量是因为指针的值是内存地址,当永久对象再次调入时,程序运行的环境已发生了变化,原先保存的指针值就失去了意义,故需对此指针值进行相应的修正,即所谓的指针变换(Swizzling).在 CASE C++中,我们是用链变量来代替指针变量从而解决永久对象的指针变换问题的.同样原因,永久对象的成员中若有静态变量,编译也不能保证其运行的正确性.因此永久对象中若含有静态变量或指针变量的声明,编译时将给出警告信息.如果一个 CLASS 定义体中没有出现 LINK 和 CONTENT 声明,则此对象类退化为与之等价的临时对象类.这时用户既可以用它来定义永久对象又可以用它来定义临时对象.如:

```
CLASS A {
        〈成员表〉;
        };
A a;
persistent A a1;
```

编译后 a 为系统的临时对象,程序运行完既消失,而 a1 为系统的永久对象,程序运行结束后仍保存在 OMS 中.

2 类定义共享

2.1 类定义共享的实现方法

所谓类定义的共享是指一个用户系统中定义的类可以被其它系统或用户方便地引用.这里的引用有 2 层意义,一是可以用它来直接定义对象,另一个是可以通过继承来生成新的

类和方法.

在CASE C++中是通过引入2个新的声明符:import和export来实现类定义共享的.其语法形式为:

import ⟨classname⟩;

export ⟨classname⟩(Mode);

其中export声明的语义是将classname类定义移出到OMS的类库中,并把该类的方法的目标码存入到方法库中.Mode同UNIX的文件权限一样是一组二进制数,它描述移出的类及方法的存取控制权限.缺省时为类的所有者可读写,其它用户可读.当用export语句移出classname类的定义体时,其所有先辈的类定义体也被隐式移出(除非类库中已经存在).

import声明的语义是从OMS的类库中移入classname类的定义.事实上就是从类库中把该类的定义复制到当前的程序中.显示移入classname类定义时,将隐含地移入该类的所有先辈的类定义和方法(不包括当前程序中已有的类定义).

使用import声明移入类定义时,要求当前用户对该类具有读权限,否则编译时将产生权限错误.

2.2 类定义共享的主要问题和解决办法

在单继承情况下,类库是一个以OBJECT为根结点所组成的一棵类树.我们把任一类A在该树中的深度记为D(A).

要共享一个类的定义,必须把与这个类相关联的所有聚集的定义体都同时移出到类库中.否则,当import进来重用时,就会发生类型未定义等错误信息.这里相关联有2层含义:一是与它的所有父类相关联;二是与它的成员体内的对象成员的类定义相关联.关联关系是一种传递关系.因此,如果对类的定义体不加任何限制规则,将会对类定义的共享带来很多副作用.例如有以下3个类定义:

```
CLASS C: public D {
        ...
      };
CLASS B {
        ...
        C c;
      };
CLASS A {
        ...
        B b;
      };
```

这时如果要共享A,我们就不得不把B、C和D一起存放到类库中.而用户在import A时可能对B、C和D并不感兴趣,但为了重用A而不得不把B、C和D一起import进来.这样将会产生两个问题:一是增加了编译的时空开销;二是当关联增加时,很难保证不发生重名错误.从而为重用带来额外的负担——需要维护类名在定义域内的有效性.当系统增大时,这是一件十分困难的工作.

为了解决这个问题,我们引入类的关联度的概念.

定义. 一个类与所有其它类相关联的程度称为该类的关联度,记为E.

因为一个类和其它类只有两种关系,即:一般和特殊;整体和部分.在CASE C++中是

通过继承和对象成员来实现这两种关系的.故有

$$E=E_s+E_i \tag{1}$$

其中 E_s 表示对象成员的关联度,E_i 表示继承关联度.

由于CASE C++中允许用缺省类名声明对象,故总可以通过增加类声明体的长度而使得 $E_s=0$.因此(1)式可变为　　$E=E_i$　　(2)

假设存在一条单继承路径 $A_0,A_1,A_2,\ldots,A_n$.其中 A_0 为叶结点,A_n 为根.且令 $E_i(A_j,A_{j+1})=1,0\leqslant j\leqslant n-1$.则有

$$E(A_0)=E_i(A_0,A_1)+E_i(A_1)=E_i(A_0,A_1)+\cdots+E_i(A_{n-1},A_n)+E_i(A_n)$$

因为 A_n 为根,显然有 $E_i(A_n)=0$.故有　　$E(A_0)=n=D(A_0)$

结论:在单继承情况下,任一类的关联度在理论上总等于它在类树中的深度.要共享的类,其关联度值越接近此值越好.

在多继承情况下,派生类和基类构成一个有向无环图(DAG).这时任一有 m 个父类,深度为 n 的结点的关联度 $E\geqslant n+\sum_1^{m-1}D(A_i)$.

由此可见,m 越大,E 增长越快.这种关联度的增加不仅影响编译效率,而且随之带来了类库维护费用的增大.所以在JB2中的CASE C++不支持多继承的类定义共享.

3　永久对象的关系描述——链

由于在永久对象中存在指针变换(Swizzling)问题,所以要想表达两个永久对象的关系就必须通过其它技术来实现.不同的系统有不同的实现方法.在CASE C++中我们是通过引入新的语言成份——链来实现的.链是两个对象间的一种关系描述.一个链从一个源对象出发,链接到一个目的对象.链作为对象的一个属性在源对象中定义,它的值则是目的对象的永久性标识.这一点是链和非永久性OOPL中的"对象指针"概念的重要区别.链的另一个特点是它可以带有自己的链属性(链属性的数据类型可为C++中的任何基本数据类型,个数不限),从而使链不仅仅可用来进行导航式查询,而且可以通过它的属性来表达两个永久对象关系的语义.如当永久对象是城市时,链可以表示两个城市之间存在的一条可达路径,而其属性可以是这条可达路径的长度或旅行的费用等.

3.1　链类型声明和链变量的定义

在CASE C++中,链类型声明有如下3种形式:

(1)LINK〈链类型名〉to〈目的类名〉{〈属性说明表〉}〈链名〉[〈正整数〉]$_{可选}$;

(2)typedef LINK to〈目的类名〉{〈属性说明表〉}〈链类型名〉;

(3)LINK〈链类型名〉to〈目的类名〉{〈属性说明表〉};

其中LINK和to是保留字,用来标识链声明.〈链类型名〉,〈目的类名〉和〈链名〉都是标识符.〈链类型名〉代表所声明的链类型,可用它来定义链变量.〈目的类名〉为链所指向的目的类的类名.〈属性声明表〉声明一组属性,属性用来描述链的语义特征.〈属性声明表〉为{〈类型名〉〈属性变量名〉;…},其中〈类型名〉为C++中的基本数据类型.

上面的(2)、(3)式仅声明了链类型.而(1)式中不仅声明了一个链类型,还定义了一个链变量.式中的[〈正整数〉]存在时,则定义了一链变量数组.

链变量声明的另一种方式为： 〈链类型名〉〈链变量名〉[〈正整数〉]$_{可选}$；

允许这种声明方式是为了符合C++的变量声明方式.

3.2 链的操作和引用

在CASE C++中对链的操作和引用主要有以下几种：

(1)链的赋值运算

〈链名1〉=〈链名2〉;让链名1也指向链名2所指的对象.

〈链名1〉=&〈对象名〉;让链名1指向一对象.

〈链名1〉=NULL;把链变量置空.

(2)删除一个链 delete〈链名〉;即释放链变量所指的对象,并将〈链名〉置空.

(3)链属性引用 〈链名〉.〈属性名〉;

(4)链的目的对象的域值引用 〈链名〉:〉〈域名〉;新引入的链运算符:〉在语义上等价于指针运算符->.

3.3 编译要解决的主要问题

引入链后,编译主要要解决以下几个问题：

第1:要能正确识别链类型、链变量声明和链表达式,并能进行语法检查和出错处理.由于链机制是永久对象模型的语言成份,因此不允许链类型和链变量声明出现在class定义体内.但链类型和链变量声明可以出现在全局定义域位置.这时链变量是一个无源的链.以后可以用赋值的方式使它指向一个静态声明的或动态创建的永久对象.

第2:由于在CASE C++中永久对象标识采用的是显示标识和可达性模型相结合的办法.所以一个永久对象的链所指向的对象也必须是永久对象.若链指向的对象是临时对象,编译时将给出不能保证该链的正确性的警告信息.为了实现动态地构造永久对象的关系图,我们引入了pnew( )函数,通过它动态产生一个永久对象.动态永久对象是无名的,事实上系统自动给它分配一个POID,当一个进程结束时,系统自动把所有的动态永久对象保存到OMS中.动态永久对象的调入对用户也是透明的.即当用户沿链导航式查询到它时,系统自动把它从OMS中调入.

第3:要保证链的完整性.即对有链指向的动态永久对象不能随便用delete语句删除.若一定要删除此永久对象,则必须先删除指向它的所有链.在CASE C++中,用delete〈链名〉来把该链名置空,同时检查该链所指的对象是否仍被其它的链所指,若无则释放此链所指向的对象,若有则保留该对象.

4 对象的内容

引入内容概念的主要目的是为了描述对象中的可变长属性.如:交互式编辑的图形、正文等信息.内容可以是一个文件,也可以是一块连续的内存块.这样使得CASE C++中的对象的粒度可以大到由多个文件所组成,小到仅含有若干个属性值.因而大大增强了对现实世界的表达能力.

4.1 内容的声明

在CAES C++中引入新的保留字CONTENT来声明内容,其语法形式为:CONTENT;内容是一个无名声明.由于内容是永久对象模型中的属性,故只允许内容声明

出现在CLASS定义体内,在此之外的其它任何地方出现,编译程序都将报错.

```
例:CLASS CTEST {
    private:
        char Projectname[NAMELEN];
        char Authername[NAMELEN];
        char filename[NAMELEN];
        CONTENT;
    };
     persistent CTEST mytest;
```

在上例中,CTEST是一个C程序测试的永久对象类的声明.而mytest是一个具有内容的永久对象.比如这个内容可以是要测试的C源程序.

4.2 内容的操作

用户对内容主要有以下几种操作:把一对象的内容调入到文件或内存中;把文件或一内存块的信息写入到一永久对象的内容中;求一内容的长度.它们的操作原型如下:

(1)char * readcontent ();

(2)int writecontent (char *p,unsigned len);

(3)int opencontent (char *filename);

(4)int writecontent (char *filename);

(5)unsigned getsize();

其中(1)式是把一对象的内容读入到内存中,并返回首地址.(2)式是把首址是p,长度为len的内存块写入到对象的内容中.(3)式是把文件filename转换为一对象的内容.(4)式是把一对象的内容写到文件中.(5)式是求一对象内容的长度.

对于上例,通常先用内容操作mytest.opencontent("test.c");把test.c源程序读入到永久对象mytest的内容中;测试工具可以把中间结果再通过操作mytest.writecontent(char * p,unsigned len)写入到OMS中.以后若要对此中间结果再加工,可通过readcontent()把它读入.

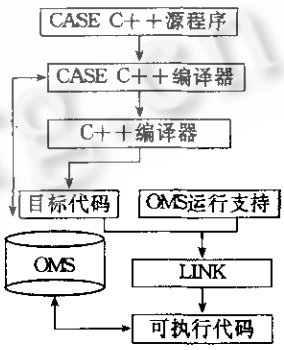

图1

5 实现方法

CASE C++语言的实现方法是首先调用CASE C++编译器将CASE C++程序编译成C++代码,然后再调用C++编译器生成可执行代码.整个过程如图1.

需要说明的是为了用户方便,CASE C++编译器不仅对新增加的语言成份进行语法分析和出错处理,而且对C++的成份也进行分析和检查.C++编译器的调用对用户是透明的.

6 结束语

CASE C++语言从设计到实现历时近3年,现已进入实用阶段.在JB2中,已有十几个集成工具由它编写,并被集成到环境中.实践证明CASE C++所提供的永久对象、类定义

的共享、链和内容等机制为用户的数据集成提供了方便而强有力的手段，使得JB2中的开发工具在整个环境中实现了紧密集成. 与此同时，我们也在考虑CASE C++的进一步完善和发展问题. 这主要集中在以下2个方面：第1，现在的类库只支持单继承，今后要在其基础上实现多继承的类库管理，从而在CASE C++中实现多继承的类定义共享. 第2，解决CASE C++与数据库的连接问题. 我们的目标是实现在CASE C++中直接调用SQL，并使它能方便地同多种数据库建立连接.

参考文献

1 The Xerox learning research group. The Smalltalk—80 System. Xerox Palo Alto Research Center, 1981.

2 Struostrup B. The C++ programming language, 2nd Edition. Addison—Wesley, Reading, MA, USA, 1991.

3 杨芙清等. 面向对象的CASE环境青鸟Ⅰ型系统的设计与实现. 中国科学(A辑), 1995.

4 杨芙清等. 永久对象存储技术研究. 电子学报, 1994, 22(8): 1～8.

5 Nguyen T A *et al*. PC++: an object—oriented database system for C++ applications. Database Systems for Advanced Applications'91. Proceedings of the Second International Symposium, 1991. 109～115.

6 Copeland G, Maier D. Making smalltalk a database system. Proc. ACM SIGMOD, 1984. 316～325.

7 Atkinson M P *et al*. PS—ALGOL: an algol with a persistent heap. ACM SIGPLAN, Nov. 1987. 24～31.

8 Cardelli L. Amber technical report. AT&T Bell Laboratories, Murray Hill, USA, 1985.

9 Deux D *et al*. The O2 system. Commun. of the ACM, 1991, **34(10)**: 34～48.

THE DESIGN AND IMPLEMENTATION OF AN OBJECT—ORIENTED PROGRAMMING LANGUAGE CASE C++ IN JADE BIRD 2 SYSTEM

Shao Weizhong Yuan Shutao Yang Fuqing

(*Department of Computer Science Beijing University Beijing* 100871)

Abstract CASE C++ is an object—oriented programming language of fully compatible with C++ and supporting persistent object, which is originally designed and implemented in the object—oriented CASE environment Jade Bird 2 System. In Jade Bird 2, it is the manipulation language of OMS (object management system) and the programming language of integrated tools. At the sametime, it is also the end users' programming language. The paper briefly introduces the background of designing CASE C++ at first. Then it focuses the discussion on the new added language components in CASE C++, such as persistent object; share of the class definition; link declaration and content of object. At last, the method of implementation is described.

Key words Object—orientation, persistent object, C++, CASE environment, object management system.