一个 Pascal 的面向对象扩充的设计与实现*

李宣东 郑国梁

(南京大学计算机科学系 南京 210093)

摘要 本文阐述了 NDOOP 的设计和实现的主要思想. NDOOP 是一个 Pascal 的面向对象扩充,是 Pascal 的超集,既充分支持面向对象程序设计,又保持了 Pascal 的原有风格.

关键词 面向对象程序设计语言,Pascal.

起初,N. Wirth 教授设计 Pascal 语言有两个非常明确的目的:(1)设计一个语言,清楚而自然地反映某些基本概念. 在这基础上,通过该语言的教学,能使程序设计成为一种系统训练. (2)定义一个语言,使得在一个可使用的计算机上,它的实现是可靠的和高效的.

然而,不容致疑:Pascal 所具有的属性已经远远超过了上述最初的目的. Pascal 语言已被公认为是定义严格、结构严谨、基于结构化程序设计原则的一种语言. 用 Pascal 语言书写的程序有许多优越性,如程序结构清晰、便于阅读、容易修改和易于移植,并且能清楚地反映数据结构和算法,非常便于交流.

随着面向对象技术的兴起,对传统通用程序语言进行面向对象扩充已成为程序设计语言的发展趋势之一. 本文介绍一个 Pascal 语言的面向对象扩充——NDOOP 的设计与实现.

NDOOP 是 Pascal 的面向对象扩充,是 Pascal 的超集,不仅能够充分支持面向对象程序设计,同时还保持了 Pascal 的原有风格和特点.

1 语 言

NDOOP 是在 Pascal 的国际标准文本[1]的基础上引入一些支持面向对象程序设计的机制所形成的 Pascal 的超集. 对 Pascal 的扩充是从指示器类型入手的. NDOOP 中的类型定义了一组值集和操作集. 指示器类型的值由单位空值以及标识值的集合组成,其中每个标识值都标识不同的变量或对象.

1.1 对 象

NDOOP 中的对象是问题域中问题对象在计算机系统中的表示. 问题域中的问题对象是现实世界中的实在对象的抽象,具有状态和行为. 现实世界的实在对象是客观实在或客观

* 本文得到国家 863 计划、国家教委博士点基金资助. 作者**李宣东**,1963 年生,1994 年博士毕业于南京大学,主要从事软件工程,面向对象程序设计语言方面的研究工作. **郑国梁**,1937 年生,教授,博士导师,主要从事软件工程方面的研究工作.

本文通讯联系人:李宣东,南京 210093,南京大学计算机科学系

本文 1994-04-11 收到,1994-11-25 定稿

实在在人脑中的反映,各自具有一定的特性,这些特性在NDOOP中称为对象的概念.

NDOOP中的对象可以看成为三元组:对象 =(接口,数据,操作).

对象的数据表示了问题对象的状态,作用于数据上的操作通过接口模拟了问题对象的行为.对象的数据又称为状态表示或内部结构;操作又称为方法;接口中所展示的方法又称为外部行为或消息.一个对象可以给别的对象发送消息以调用其相应的方法,又可以接收消息调用自身相应的方法,以完成一定的计算任务,改变自身的状态.因此,对象又可以定义为具有通信和独立计算能力的封装体.

NDOOP的对象由指示器变量动态地标识,可以随时通过动态分配过程new(p)创建或通过动态分配过程dispose(p)取消.

1.2 类

NDOOP中的类定义了一组具有相同概念、内部结构和外部行为的对象.类由实例变量集和方法集构成,是创建对象的模板,对象则是类的实例.

一个类的定义由两部分组成:说明部分和实现部分.类定义的说明部分给出该类的概念、不变式、实例用户接口和子类用户接口,类定义的实现部分定义了该类的实例变量和方法:

```
CLASS classname;
  PROPERTY     {…};
  INVARIANT    {…};
  CIF          (…);
  IIF          (…);
  INHERIT      … ;
  REDEFINE     … ;
  VAR          … ;
  METHOD       … ;
END classname.
```

类的概念是指其实例对象所具有的概念;类的实例用户接口(CIF)展示了其实例对象的外部行为;类的子类用接口(IIF)展示了其可以被子类继承的实例变量和方法;定义在类的实例变量上的类的不变式和类的实例变量合在一起表示了类的实例对象的内部结构;类的方法实现了其实例对象的外部行为.

类的概念由类名集合组成:类 A 的概念为 $\{A_1,A_2,\cdots,A_n\}$(A_i 为类名,$i=1,2,\cdots,n$)表示类 A 的概念包含了类 $A_i(i=1,2,\cdots,n)$的概念.

类的不变式描述了其实例对象的状态在生成期间必须满足的条件.类的不变式由类名集合组成:类 A 的不变式为 $\{A_1,A_2,\cdots,A_n\}$(A_i 为类名,$i=1,2,\cdots,n$)表示类 A 的不变式包含了类 $A_i(i=1,2,\cdots,n)$的不变式.

类的实例用户接口(*CIF*)由一组方法的说明构成.方法的说明包括型构和规范两部分.方法的规范可以采用定义在类的实例变量集上的前后置条件加以描述:$\{Pre\}m(P)\{Post\}$.前置条件 $Pre=P(s,P)$描述了方法执行前该类的实例对象的状态和方法参数所应满足的条件;后置条件 $Post=Q(s,s_0,P,r)$描述了方法执行后该类的实例对象的状态和方法结果所应满足的条件,这里 s 表示对象的当前状态;s_0表示方法执行前对象的状态;p 表示方法的参

数;r 表示方法的结果. 一般来说,描述一个类的不变式或方法的规范需要至少具有一阶谓词逻辑演算的描述能力的语言[2],由于目前程序设计语言尚未具备这样的描述能力,因此,这里类的不变式和方法的规范均用类名集合表示.

类 A 中方法 m 的规范可以在类 A 中定义,也可以根据其它类中定义的具有相同型构的方法 m 的规范加以定义. 因此,类 A 中方法 m 的规范可能有3种表示形式:$\{A\}$,$\{B\}$和$\{A,B\}$(A,B 为类名,$A\neq B$).

类 A 中方法 m 的规范为$\{A\}$表示类 A 中方法 m 的规范由类 A 自己定义;

类 A 中方法 m 的规范为$\{B\}$($A\neq B$)表示类 A 拥有类 B 的全部实例变量,并且类 A 中方法 m 的规范和类 B 中方法 m 的规范相同:若类 A 中方法 m 的规范为$\{Pre_mA\}m(P)\{Post_mA\}$,类 B 中方法 m 的规范为$\{Pre_mB\}m(P)\{Post_mB\}$,则$(Pre_mB\rightarrow Pre_mA)\wedge(Post_mA\equiv Post_mB)$;

类 A 中方法 m 的规范为$\{A,B\}$($A\neq B$)表示类 A 拥有类 B 的全部实例变量,并且类 A 中方法 m 除具有类 B 中方法 m 所具有的功能之外,还有其他功能:若类 A 中方法 m 的规范为$\{Pre_mA\}m(P)\{Post_mA\}$,类 B 中方法 m 的规范为$\{Pre_mB\}m(P)\{Post_mB\}$,则$(Pre_mB\rightarrow Pre_mA)\wedge(Post_mA\rightarrow Post_mB)$.

以上类的概念、不变式和方法的规范均由类名集合表示,这些类名构成的集合本身就是或可以转换为如下形式的类名集合:

$$\{A_1,A_2,\cdots,A_n\},$$

这里类 $A_i(i=1,2,\cdots,n)$中的概念、不变式或方法 m 的规范为$\{A_i\}$或$\{A_i,A_j\}(i,j=1,2,\cdots,n,i\neq j)$,这种形式的类名集合称为类的概念、不变式或方法的规范的完全表示形式.

方法的型构是指方法名和方法可能具有的参数和结果类型.

例如,在 NDOOP 程序中类 Point 的定义如下:

```
CLASS Point;
  PROPERTY {Point};
  CIF ( x():INTEGER{Point};
        y():INTEGER{Point};
        moveby(P:Point){Point} );
  IIF ( xpos,ypos:INTEGER;
        x():INTEGER{Point};
        y():INTEGER{Point};
        moveby(P:Point){Point};
  INVARIANT {Point};
  VAR xpos,ypos:INTEGER;
  METHOD
     x():INTEGER{Point}; BEGIN x:=xpos END;
     y():INTEGER{Point}; BEGIN y:=ypos END;
     moveby(P:Point) {Point};
        BEGIN xpos:=xpos+P.x
              ypos:=ypos+P.y
        END
  END Point.
```

1.3 对象类型

在NDOOP中,对象按概念、内部结构和外部行为分类,对象类型和类合一.因此类作为对象类型是一组共有该类的实例对象所具有的概念、内部结构和外部行为的对象构成的集合.

NDOOP中,对象由指示器变量标识.类C作为对象类型是一组对象构成的集合,设该集合为S,则$i:C$说明了一个标识集合S中的元素(对象)的指示器变量i.向指示器变量i所标识的对象发送消息m可以表示成$i\cdot m$.$i\cdot m$可以是语句,也可以是表达式.

类(对象类型)A是类(对象类型)B的子类型(记为$A<B$)的充要条件是:

(1)类A的概念的完全表示形式包含类B的概念的完全表示形式;

(2)类A的不变式的完全表示形式包含类B的不变式的完全表示形式;

(3)对类B的实例用户接口中出现的每一个方法m,在类A的实例用户接口中存在相同型构的方法m,并且类A中的方法m的规范的完全表示形式包含类B中的方法m的规范的完全表示形式.

例如,有如下定义的类A,B,C:

```
CLASS A;
 PROPERTY {A}
 CIF (ma1{A}, ma2{A});
 IIF ( va1:Ta1, ma1{A}, ma2{A} )
 INVARIANT {A};
 VAR va1:Ta1;
 METHOD
    ma1{A}; BEGIN … END;
    ma2{A}; BEGIN … END;
 END A;
CLASS B;
 PROPERTY {A,B}
 CIF ( ma1{A}, ma2{A,B}, mb1{B} );
 IIF ( va1:Ta1, vb1:Tb1, ma1{A}, ma2{A,B}, mb1{B} )
 INVARIANT {A,B};
 VAR va1:Ta1; vb1:Tb1;
 METHOD
    ma1{A}; BEGIN … END;
    ma2{A,B}; BEGIN … END;
    mb1{B}; BEGIN … END;
 END B;
CLASS C;
 PROPERTY {B}
 CIF ( ma1{A}, ma2{B}, mb1{B, C} );
 IIF ( va1:Ta1, vb1:Tb1, ma1{A}, ma2{B}, mb1{B,C} )
 INVARIANT {B};
 VAR va1:Ta1; vb1:Tb1;
 METHOD
```

```
    ma1{A}; BEGIN … END;
    ma2{B}; BEGIN … END;
    mb1{B,C}; BEGIN … END;
END C;
```

下表给出了类 A、B、C 的概念、不变式和实例用户接口中所展示的方法规范的完全表示形式:

类名	*PROPERTY*	*INVARIANT*	*ma*1的规范	*ma*2的规范	*mb*1的规范
A	$\{A\}$	$\{A\}$	$\{A\}$	$\{A\}$	
B	$\{A,B\}$	$\{A,B\}$	$\{A\}$	$\{A,B\}$	$\{B\}$
C	$\{A,B\}$	$\{A,B\}$	$\{A\}$	$\{A,B\}$	$\{B,C\}$

由上表知,$B<A$,$C<A$,$C<B$.

NDOOP 中定义类间的子类型关系是为了支持子类型多态. 若有说明:i:A; j:B(A,B 为类名),并且 $A<B$,则赋值语句 $j:=i$ 及相应的实、形参数匹配是合法的.

NDOOP 允许在类定义的实现部分只给出方法的型构和规范,这样定义的类称为抽象类:

```
ABSTRACT CLASS classname;
        PROPERTY {…};
        INVARIANT {…};
        CIF {…};
        IIF {…};
        INHERIT … ;
        REDEFINE … ;
        VAR … ;
        METHOD
            所有或部分方法可以只给出型构和规范,不给出实现
END classname.
```

抽象类只能用作对象类型,不能作为创建对象的模板. 抽象类可以看成是抽象数据类型的规格说明,可作为从设计到实现的一种自然过渡.

1.4 继　承

NDOOP 中,继承只作为代码复用机制,子类通过继承拥有全部父类的子类用户接口中展示的实例变量,全部或部分拥有父类的子类用户接口中展示的方法,并且保持父类的不变式.

子类可以重定义其继承的父类方法以改变方法的实现、规范或型构(不包括名). 但是,在子类中未被重定义的父类方法必须保持其在父类中的规范. 设类 A 是类 B 的子类,类 B 中的方法 m 被类 A 所继承,则类 A 中的方法 m 的规范按如下规则定义:

(1)若类 A 中没有重定义方法 m,则类 A 中方法 m 的规范为$\{B\}$;

(2)若类 A 中重定义方法 m 以改变其实现,则类 A 中方法 m 的规范为$\{B\}$;

(3)若类 A 中重定义方法 m 以扩充其规范,则类 A 中方法 m 的规范为$\{A,B\}$;

(4)若类 A 中重定义方法 m 以改变其规范或型构(不包括名),则类 A 中方法 m 的规范的完全表示形式不包含类 B 中方法 m 的完全表示形式.

类 A 重定义方法 m 以扩充其规范是指重定义后方法 m 除具有类 B 中的方法 m 的规范所表示的功能外，还具有新的功能.

NDOOP 允许子类通过继承拥有多个父类的实例变量和方法. 子类的多个父类间可能存在名冲突问题，从而使得子类可能出现如下情况：

(1)两个或两个以上的实例变量同名，但其类型不同；

(2)两个或两个以上的实例变量同名，类型也相同；

(3)两个或两个以上的方法的名相同，但其参数和结果类型不同；

(4)两个或两个以上的方法的名相同，其参数和结果类型也相同.

上述4种情况在类的定义中出现都是不合法的，因此，当情况(1)出现时，子类不能通过继承拥有这些父类的实例变量和方法；当情况(2)出现时子类或者在保持这些父类的不变式的条件下将这些实例变量合一，或者不能继承这些父类；当情况(3)出现时，子类或者只能通过继承拥有这些同名、但参数和结果类型不同的方法中的一个，或者不能继承这些父类；当情况(4)出现时，子类或者只能通过继承拥有这些同名且同参数和结果类型的方法中的一个，或者将这些方法合一并且重定义，或者不能继承这些父类.

NDOOP 允许在定义类的方法时调用该类的方法. 为了使方法调用和消息发送相一致，引入伪变量 *self* 用以表示正在执行该类的方法的对象，这样在定义类的方法时调用该类的方法可以表示为向 *self* 发送消息. 这里 *self* 不能被赋值，也不能用作实在参数.

当子类重定义了父类的方法 m 后，一般情况下，子类中的方法 m 过载了父类中的方法 m. 这里允许子类直接调用父类中的方法 m，采用的形式是 *super*. m . *super* 也是一个伪变量. 当子类拥有多个父类时，伪变量 *super* 和父类名合起来使用：*super*_*SC*. m(*SC* 为类名).

NDOOP 中继承机制的语义用类似于 Smalltalk－80[3]中的方法查询算法可描述如下：

消息接收者在接到消息后，在自己所属类中查询与消息匹配的方法. 若找不到，则继续在超类中查找；若再找不到，则继续在超类的超类中查找. 如此一直进行下去，直至找到为止.

若消息的接收者为 *self*，设在包含 *self* 的方法所在的类中，与消息匹配的方法的规范的完全表示形式为 S(S 为类名集合)，则从发送者所属类中开始、沿着发送者为调用包含 *self* 的方法所形成的、从发送者所属类到包含 *self* 的方法所在类的路径，查询规范的完全表示形式为 S 且与消息匹配的方法.

若消息被发送给 *super*，则查询从包含 *super* 的方法所在类的直接超类开始进行.

2 实 现

对现有的程序设计语言进行面向对象扩充，其实现一般有两种方法：一是重做语言的编译器；而是用被扩充的语言实现扩充后的语言. 语言 DRAGON[4]、C++[5]的早期版本都采用了后一种方法. 考虑到后一种方法简单、易行、便于产生扩充后的语言的原型以便进一步修改和验证，这里采用后一种方法实现 NDOOP，即将 NDOOP 书写的程序转换为标准 Pascal书写的程序，然后用 Pascal 的实现系统实现之.

3 小 结

NDOOP 作为 Pascal 的面向对象扩充，具有下述特点：(1)充分支持面向对象程序设计；(2)保持了 Pascal 的原有风格；(3)继承仅作为代码复用机制，具有较大的灵活性；(4)子类型关系和继承分离，类间子类型关系由编译程序自动确认，提高了程序的可靠性；(5)类的规范(接口)与实现分离，类成为真正的可复用模块，从而程序(类)的可靠性、可重用性和易维护性得到提高.

根据上述特点，NDOOP 和现有面向对象程序设计语言相比在很多方面具有优势. 美国 Borland 国际公司推出的 Turbo Pascal 自5.5版本起，扩充了支持面向对象程序设计的设施. NDOOP 与之相比在支持面向对象程序设计方面更充分、完善.

参考文献

1 崔桐豹，程虎译. 计算机程序设计语言 Pascal(国际标准文本). 国防工业出版社，1988.

2 America P. A parallel object—oriented language with Inheritance and Subtyping. ECOOP/OOPSLA'90, 1990.

3 Gooldberg A, Robson D. Smalltalk—80: the language and its implementation. Addison—Wesley, 1983.

4 Atkinson C. Object—oriented reuse, concurrence and distribution. Addison—Wesley, 1991.

5 Stroustrup B. The programming language C++. Addison—Wesley, 1986.

DEVELOPING AN OBJECT—ORIENTED EXTENSION OF PASCAL

Li Xuandong　Zheng Guoliang

(*Department of Computer Science　Nanjing University　Nanjing*　210093)

Abstract　The idea in design and implementation of NDOOP, which is an object—oriented extension of Pascal, is introduced in this paper. NDOOP is a superset of Pascal. It not only supports sufficiently object—oriented programming but also follows in the spirit of Pascal.

Key words　Object—oriented programming language, Pascal.