文章编号:1001 - 9081(2005)04 - 0796 - 03

面向对象的契约式程序设计

林佳一,刘　进,何克清
(武汉大学 计算机软件工程国家重点实验室,湖北 武汉 430072)
(a2002ljy@ sina. com)

摘　要:运用行为子类型及扩充行为子类型的概念,通过对一个 Java 实例地剖析,讨论了在面向对象的契约式程序设计中如何撰写契约,以保持面向对象的单个继承和多重继承的特性问题,并证明了这一方法的有效性。最后探讨了动态环境下违反契约时的责任归咎,展望了契约思想在软件开发中的运用前景。

关键词:行为子类型;契约;前置条件;后置条件;不变式
中图分类号: TP311.11　　**文献标识码**:A

Object-oriented programming by contract

LIN Jia-yi, LIU Jin, HE Ke-qing
(*The State Key laboratory of Software Engineering, Wuhan University, Wuhan Hubei* 430072, *China*)

Abstract: A notion of behavior subtyping and its extension was introduced. By analysing an example of Java, how to design in object-oriented programming by contract was discussed in order to keep the features of single and multiple inheritance. The effectivity of the method was demonstrated. Finally, how to assign blame when contracts were violated in dynamic enviroment was discussed. The perspective of contract used in software development was outlooked.

Key words: behavior subtyping; contract; precondition; postcondition; invariant

0　引言

Bertrand Meyer 提出的契约式程序设计是一种改善软件工程质量的有效手段。使用契约进行程序设计有助于提高程序设计的可靠性和效率。

契约包括前置条件、后置条件和不变式,它们通常以布尔表达式的形式在程序中出现。在面向对象的程序设计中,前置条件和后置条件都针对方法,分别规定了在调用该方法前后必须为真的条件。不变式针对整个类,它规定了该类任何实例调用任何方法时都必须为真的条件。在结构化的程序设计中,把前置、后置条件、不变式插入到过程中,通过在运行时检查这些条件是否满足来保证程序地正确执行。如果一个过程的前置条件不成立,过程将无法执行。如果它的后置条件不成立,表示过程的结果不正确。不变式描述了过程运行后不会发生改变的特性。

面向对象的继承性、多态性和封装性大大促进了软件的重用和扩充,但同时增加了契约在面向对象程序设计中应用的复杂性。在面向对象的程序设计中,子类的方法是可以覆盖其父类的方法,而在类的方法中加入契约后,类的继承允许子类的对象替换父类的对象,那么子类方法的契约与父类方法的契约之间必须有一个关系以确保子类的对象能够正确地替换其父类的对象,因此仅仅检查子类方法的前置、后置条件无法保证程序能继续正确保持其面向对象方法中的单个继承和多重继承特性。

1　行为子类型及其扩充

类作为面向对象程序设计的基本元素,用来刻画该类对象的属性和行为,通过继承可以在已有类的基础上构造新类,其本质是实现在层次结构上类的共享。行为子类型为形成健壮的类层次结构提供了一个理论基础,使得类的各层作为一个良好结构整体出现在各种环境中,真正实现了面向对象方法中的多态性,即子类对象能够替换父类对象。

1.1　行为子类型

从抽象角度和外部特征来理解,最好的方式是将类看作类型的实现。类实际是类型的内部设计,它实现了类型的外部特征。许多面向对象语言都有类型机制,且通常以类作为类型,以子类作为子类型。美国科学家 Liskov、Wing 和 Meyer 提出了在面向对象的世界里,与行为规范相关的类型和子类型概念以更加明确的表示面向对象方法中这种继承关系中的替换性。

行为子类型定义:如果子类型的对象可以替换程序中类型的出现,而对程序的行为没有影响,那么这个子类型是一个行为子类型。

从语义上讲,对于行为子类型,一个类型的前置条件蕴涵它的子类型的前置条件;一个子类型的后置条件蕴涵这个类型的后置条件。

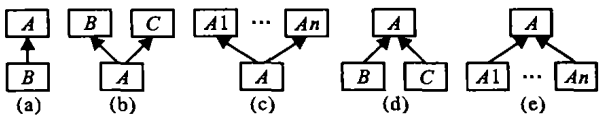

图 1　行为子类型层次结构

图 1 是行为子类型的层次结构,图 1(a)描述了仅涉及两个类的最简单的层次结构。图 1(a)中有两个类 A、B, B 是 A 的子类。两个类都有方法 m,类 B 中的方法 m 覆盖了类 A 中的方法 m。p、q 分别表示方法 m 的前置、后置条件。x 表示方法 m 的输入。如果 B 是 A 的行为子类型,则应该满足条件:

收稿日期:2004 - 09 - 02;修订日期:2004 - 11 - 17　基金项目:国家自然科学基金资助项目(60373086)
作者简介:林佳一(1975 -),女,湖南人,硕士研究生,主要研究方向:软件工程;　刘进(1977 -),男,湖北人,博士研究生,主要研究方向:软件工程、元建模和本体应用;　何克清(1947 -),男,湖北人,教授,博士生导师,主要研究方向:软件工程、软件平台.

$\forall x(p^A(x) \rightarrow p^B(x))$　　$\forall x(q^B(x) \rightarrow q^A(x))$

也就是对于任何输入 x,A 的前置条件 $p^A(x)$ 蕴涵 B 的前置条件 $p^B(x)$,同时 B 的后置条件 $q^B(x)$ 蕴涵 A 的后置条件 $q^A(x)$。

1.2 扩充的行为子类型

在两层的面向对象的层次结构中,继承关系主要以图 1(a)~(e)的结构形式表现出来。前面已经分析了图 1(a)中层次结构的行为子类型条件,下面通过分别考虑每个行为子类型关系,将行为子类型的概念扩充,得到了一个涉及多个类的扩充行为子类型的概念。

图 1(b)~(e)给出了一个扩充行为子类型的结构描述。图中 A、B、C、$A1$、…、An 是类,它们都有方法 m,及各自定义的前置、后置条件 p、q。带箭头的直线表示继承关系,箭头所指方向为父类。

图 1(b)中,A 是 B 的行为子类型,同时 A 又是 C 的行为子类型。那么对于任何输入 x,前置条件要满足:

$$\forall x((p^B(x) \rightarrow p^A(x)) \wedge (p^C(x) \rightarrow p^A(x))) \quad (1)$$

后置条件要满足:

$$\forall x((q^A(x) \rightarrow q^B(x)) \wedge (q^A(x) \rightarrow q^C(x))) \quad (2)$$

将式(1)、(2)经恒等变形后得到简化形式:前置条件要满足 $\forall x((p^B(x) \vee p^C(x)) \rightarrow p^A(x))$,后置条件要满足 $\forall x(q^A(x) \rightarrow (q^B(x) \wedge q^C(x)))$。

图 1(c)中,将图 1(b) A 有两个父类的情况推广到有 n 个父类的更一般的情况,进一步扩充行为子类型概念。采用简化形式:前置条件要满足 $\forall x((p^{A1}(x) \vee p^{A2}(x) \vee \cdots \vee p^{An}(x)) \rightarrow p^A(x))$,后置条件要满足 $\forall x(q^A(x) \rightarrow (q^{A1}(x) \wedge q^{A2}(x) \wedge \cdots \wedge q^{An}(x)))$。

图 1(d)中,B 是 A 的行为子类型,同时又 C 是 A 的行为子类型。那么对于任何输入 x,前置条件要满足:

$$\forall x((p^A(x) \rightarrow p^B(x)) \wedge (p^A(x) \rightarrow p^C(x))) \quad (3)$$

后置条件要满足:

$$\forall x((q^B(x) \rightarrow q^A(x)) \wedge (q^C(x)) \rightarrow q^A(x)) \quad (4)$$

将式(3)、(4)经恒等变形后得到简化形式:前置条件要满足 $(p^A(x) \rightarrow (p^B(x) \wedge p^C(x)))$,后置条件要满足 $\forall x((q^B(x) \vee q^C(x)) \rightarrow q^A(x))$。

图 1(e)中,将图 1(d)中 A 有两个子类的情况推广到有 n 个子类的更一般的情况,进一步扩充行为子类型概念。采用简化形式:前置条件要满足 $\forall x(p^A(x) \rightarrow (p^{A1}(x) \wedge p^{A2}(x) \wedge \cdots \wedge p^{An}(x)))$,后置条件要满足 $\forall x((q^{A1}(x) \vee q^{A2}(x) \vee \cdots \vee q^{An}(x)) \rightarrow q^A(x))$。

在面向对象的结构层次图中,任何一种复杂的关系都可以由以上的几种基本的关系嵌套组合而成。如果图中的每一个类和它的父类都满足以上所定义的行为子类型的定义,那么可以说这个类的层次结构是健壮的,它能够保证类对象的替换正常进行。

2 行为子类型在契约式程序设计中的研究

在面向对象程序设计中,为一个子类撰写契约时,由于可以在子类中重新定义这个父类的某些特性,而这可能需要在子类中修改从父类继承来的这些特性的契约。因此,契约的修改必须考虑到子类和父类之间的继承关系,换而言之,子类中的契约是否能替换父类的契约而不使两者之间出现冲突。

为了解决这一问题,这里使用了行为子类型和扩充的行为子类型概念,将它们用于指导在面向对象方法中的契约式程序设计。通过对以下一个采用契约的 Java 程序的分析,可以更加清楚地理解行为子类型和扩充的行为子类型是如何用来实现在单个继承、多重继承中前置、后置条件的撰写,从而能够很好的重定义一个子类的契约,使之遵守从它父类继承的契约并能够直接替换其父类的契约。

程序段 1:

```
interface I
  { int m(int a, int b) { }
    @pre (a>5)
    @post(b > 1)
    …
  }
class A implements I, J
  { int m(int a, int b) {b = a * a}
    @pre(a>2)
    @post(b>4)
    …
    @invariant (C>15)
  }
interface J
  { int m(int a, int b) { }
    @pre (a>3)
    @post (b> 1)
    …
  }
class B extends A
  { int m(int a, int b) {b=5 * a}
    @pre(a>1)
    @post(b>5)
    …
    @invarriant ((C>15) &&(C>30))
  }
```

2.1 行为子类型与单个继承

程序段 1 中定义了两个类 A、B,其中 B 是 A 的子类。它们都有相同的方法 m,B 的 m 方法覆盖了它父类 A 的方法 m。@pre、@post 是前置、后置条件的关键字。B 的 m 方法的前置条件要求输入参数 $a > 1$,而它的父类 A 的 m 方法的前置条件要求输入参数 $a > 2$。

对于方法 m 的前置条件,当输入参数 $a \geq 3$ 时,满足父类 A 的对象的方法 m 的前置条件 $a > 2$,方法 m 可以正常调用,同时也满足子类 B 的对象方法 m 前置条件 $a > 1$,因此子类 B 的对象方法 m 也能够正常调用。如果把 B 类中方法 m 的前置条件修改为 $a > 4$,那么当输入参数 a 为 3 时,满足父类 A 方法 m 的前置条件 $a > 2$,但不满足子类 B 方法 m 的前置条件 $a > 1$。因此子类 B 中的 m 方法不能执行,而父类 A 中的方法 m 能够执行,这种冲突使得子类对象无法替换父类对象的出现。

方法执行完后,就要对它的后置条件进行检验。子类 B 中方法 m 的后置条件要求 $b > 5$,比父类 A 中方法 m 的后置条件 $b > 4$ 的约束更强,可见当子类 B 中的方法 m 能够正确执行完毕时,其父类 A 中的方法 m 也一定能够正确执行。这个后置条件关系的设置说明了一旦子类对象替换了父类对象,那么子类方法的正常运行就能够确保父类方法的正常运行。

通过以上分析可以说明,要实现类继承关系的契约,子类的前置条件要比父类的前置条件制约性要弱,而子类的后置条件的制约性要强于父类的后置条件。换言之,子类和父类方法的前置、后置条件从语义上要满足行为子类型的规范。B 是 A 的子类型,因为前置条件关系式 $(a > 2) \rightarrow (a > 1)$ 和后置条件关系式 $(b > 5) \rightarrow (b > 4)$ 恒成立。

2.2 扩充行为子类型与多重继承

Java 不象C++有多重继承的概念,它是通过类实现多个

接口来体现多重继承的思想。接口中的方法只有方法名没有方法体,由实现该接口的类来实现方法体。接口的方法中可以加入前置、后置条件,对于实现多个接口的类,类方法与多个接口方法中的前置、后置条件也存在着特定关系以体现多重继承的特点。根据前面讨论的扩充行为子类型概念,在整合多接口的方法契约编写时,对上层接口中方法的前置、后置条件的定义要十分慎重。

在程序段 1 中,类 A 的方法 m 前置条件要求输入参数 $a > 2$,方法体中将 a 平方后赋值给 b,定义 $b > 4$ 成为方法 m 的后置条件。A 实现的两个接口 I、J 中方法 m 的前置条件分别为 $a > 5$、$a > 3$。

如果仅从数学运算结果的角度考虑,常常把接口 I、J 中 m 的后置条件分别设为 $b > 25$、$b > 9$。但这样做的直接后果会导致,当类 A 的方法 m 中输入参数 $a = 4$ 时,方法 m 被调用执行完后会出现错误,因为类 A 同时继承了接口 I、J 中方法 m 的前置、后置条件,而方法 m 执行完后 $b = 16$,这使得接口 I 中方法 m 的后置条件 $b > 25$ 不成立。即 $(b > 4) \rightarrow (b > 25)$ 为假,和 $(b > 4) \rightarrow (b > 9)$ 进行与运算后还是为假。所以为了保证类 A 能同时实现两个接口而又不受到契约的影响,将 I、J 中方法 m 的后置条件弱化,在这里都设为 $b > 1$。这样就满足了 A 分别是 I 和 J 的行为子类型的扩充行为子类型条件。

3 不变式与继承

与前面讨论的前置、后置条件不同,不变式所体现的概念是贯穿在整个类范畴中,在类对象的整个生命周期中它都保持恒定不变的特性。不变式出现在类定义的尾部,它的作用范围是类中的所有方法。对于类的继承关系,Mitchell 认为,子类中不变式要遵循它的父类的不变式,同时子类可以在不变式中增加自己的不变式,新增加的不变式和被继承的不变式进行与运算,从而强化了被继承的不变式。

在程序段 1 中,不变式以关键字@ invariant 开头,有两个类 A 和 B,子类 B 继承了父类 A 所有方法和契约。父类 A 中的不变式要求 $C > 15$,而对于子类 B 中的不变式,它首先要求 $C > 15$ 同时还必须要求 $C > 30$。不变式加强了限制条件,只要它当中的不变式成立,其父类中的不变式总是成立。如果用 V_A 表示父类 A 中的不变式,$V_{A'}$ 表示子类 B 中从父类 A 中继承而来的不变式,V_B 表示子类 B 新增的不变式,子类 B 的不变式为 $V_{A'} \wedge V_B$,则 $V_{A'} \wedge V_B \rightarrow V_A$ 是一个逻辑恒等式。这种在继承关系下不变式的定义从语义上与后置条件所要求满足的关系相同,只是不变式针对类中所有的方法,后置条件仅限于一个方法,从这种意义上,可以把不变式看成是对后置条件的一种扩展。

4 在违反契约的情况下责任归咎

前面讨论了面向对象的程序设计中是如何在行为子类型定义的指导下添加前置、后置条件以及如何构造子类的不变式来保持继承性,程序员在设计程序时应该遵循这些规则。然而,以上这些都是在静态环境下的契约的设计,下面将分析在动态环境下契约的使用。

在结构化的程序设计中,前置、后置条件分别置于过程的首尾。当前置条件不成立时,过程不被执行,将责任归咎于过程前面调用它的代码;当后置条件不成立时,过程虽然执行完毕,但过程在执行中存在错误,因此将责任归咎于该过程。

而面向对象的程序在运行过程中的执行路径与结构化的程序相比要复杂得多,在实际情况中,有时程序员会出现一些疏忽,以至产生出错误的契约。这些错误的契约直接导致了程序的非正常执行,有必要对造成这些错误的契约进行分析,找出责任的归咎者。这样做能对面向对象技术中契约的动态使用过程有一个更深入的了解,同时对构造一个正确的契约编译工具也是大有裨益的。

程序段 2:

```
程序员 A
interface I
  { void m(int a){}
    @pre{a>0}
  }
interface J extends I
  { void m(int a){}
    @pre {a>15}
  }
程序员 B
class C implements J
  { void m(int a){...}
    @pre {a>15}
    public static void
    main(string argv[ ])
      { I i=new C();
        i.m(6)
      }
  }
```

如果有两个不同的程序员 A 和 B,分别写了程序 2 的两个不同部分。首先,程序员 B 的 main 方法创建了 C 的实例 i,其类型为 I。调用 i 的 m 方法实际参数为 6。符合 I 的前置条件 $a > 0$。但对于 J 的前置条件 $a > 15$,这是一个不合法的输入。行为子类型条件是,如果 J 能够替换 I,它才能够成为 I 的行为子类型。然而,这里条件不成立,J 接受的参数比 I 少,因此程序员 A 要求 J 扩展 I 是错的。当程序员 B 的方法调用失败时,违反契约的责任应归咎于程序员 A 对 J 接口定义。

这个例子说明面向对象的契约检查应该针对四类错误:前置条件违反,后置条件违反,不变式违反和继承错误。继承错误是指因为前置条件的继承或后置条件的继承没有正确的构造,子类或扩展的接口不是一个行为子类型。继承错误将面向对象的契约检查和过程中的契约检查区分开来。

5 结语

在面向对象的世界中采用契约式程序设计方法,不仅使程序设计更加系统化、更清楚、更简单,而且还可以帮助开发者能更好的理解代码和测试工作的到位。

今后,契约式编程思想不仅只用在代码的设计阶段,而且也能够用在项目开发的其他阶段,比如,分析阶段,项目管理阶段。同时结合 UML,OCL 等各种建模方法和语言以及设计模式、软件框架等这些软件设计的思想,以便在尽可能小的代价下开发出可靠性出众的软件系统。

参考文献:

[1] LISKOV BH, WING JM. A behavioral notion of subtyping[J]. ACM Transactions on Programming Languages and Systems (TOPLAS), 1994, 16(6): 1811 - 1841.

[2] MEYER B. Object - oriented Software Construction [M]. Prentice Hall, 1988.

[3] FINDLER RB, LATENDRESS M, FELLEISEN M. Behavioral contracts and behavioral subtyping[A]. Proceedings of the 8th European software engineering conference[C]. 2001.

[4] 孟岩. Design by Comtract 原则与实践[M]. 北京: 人民邮电出版社, 2003.