计算机科学 1997Vol.24 №.5

面向对象语言的动态特性研究

On the Dynamic Features of Object-Oriented Programming Languages

陆 陪 于大川 吕 建

(南京大学计算机软件研究所 计算机软件新技术国家重点实验室 南京 210093)

摘 要 One of the most important characters of object-oriented programming languages is that they have dynamic features. In this paper, polymorphism, dynamic linking, dynamic parametrism, untyped mechanism and dynamic inheritance are analyzed respectively. Then, a new idea about providing more powerful dynamic mechanisms in the level of language is given. At last, advantages and disadvantages, utility and implementation of these dynamic features are summarized.

关键词 Dynamic features, Polymorphism, Dynamic linking, Dynamic parametrism, Dynamic inheritance.

1. 引言

随着面向对象软件开发技术的发展,越来越多的软件开发者正在使用面向对象程序设计语言[1,10]。面向对象语言的重要特征之一是其动态特性。一般而言,静态与编译阶段相联系,而动态则是与运行阶段相关联的。我们这里的动态特性,就是指在运行时刻确定或改变某些语言成分的类型(或类)信息的特性。例如,几乎每一种面向对象程序设计语言,都提供了类间的继承机制,并允许子类的对象可以作为父类的对象,那么就具有动态绑定的特性。其实,动态绑定已经成为面向对象语言的主要特点之

·著作性:应当象 HTML 那样具有特定的著作工具,用户在设计三维虚拟环境时并不需要编程或手工键入 VRML 代码。

·接口:必须有供其他应用程序访问或修改三维场景的接口。

·多用户性:可供不同用户同时访问同一个三维场景。

·可伸缩性:应该允许伸缩许多小型的场景来构造大型场景。

·兼容性:VRML 是一种用于描述三维虚拟环境的标准,因此从根本上对 VRML 的语法进行修改是用户不能接受的。VRML 的后继版本应该尽可能地与以前的版本相兼容。

小结 VRML 是 Internet 网上用于发布三维虚拟场景的一种强有力的描述性语言。由于 VRML 为建立三维场景开辟了一条崭新的路子,目前正被越来越多的用户所接受,非常有可能很快成为在 Internet 网上适用于三维虚拟场景的标准交换格式。目前 VRML 尚只能浏览静态的三维场景,但是由于 VRML 的开放性以及众多研究开发人员对未来 VRML 规范所提出的合理建议,相信在不久的将来 VRML 将集成更为强有力的功能使之更好地用于描述和交互三维虚拟场景。

参考文献

[1] 赵彧、向辉等,HTML 文档规范及其应用.全国第五届多媒体学术会议论文集

[2] QvLib-A VRML Parsing Library, [www]http://vrml.wired.com/vrml.tech/qv.html

[3] G. Bell, A. Parisi and M. Pesce, The Virtual Reality Modeling Language, Version 1.0 Specification, [www]http://vrml.wired.com/vrml.tech/

[4] S. Matsuba, B. Roehl, Special Edition Using VRML, Que Corporation, 1996

陆陪 硕士生;于大川 硕士生;吕建 教授,博士生导师,计算机软件新技术国家重点实验室主任,主要研究方向为:软件自动化,并行面向对象程序的形式化方法和面向对象语言与环境。

一，使得对于一个对象可以根据其运行时刻不同的实际对象内容，来自动决定不同的方法调用。显然，动态特性有利于增强语言的描述能力和使用灵活性。

本文按动态程度逐渐增强的次序，分析面向对象程序设计语言几种典型的动态特性。提出了关于动态性更强的面向对象语言的设想，并初步从环境和语言两个方面作了实现方面的探讨。最后，对动态特性的优缺点、适用性和实现进行了总结。

2. 动态特性分析

2.1 多态及其动态特性

面向对象程序设计语言的动态特性往往与其多态密切相关，所以我们先仔细地探讨一下多态，

多态这一概念最早起源于 C. Strachey 的多态函数概念，用于刻画其参数可以取多种类型的函数。从广义上来说，多态就是指论域中的一个元素有多种解释，具体到面向对象语言，主要是指下面三种多态[2]。

(1)一名用多。这一语言现象其实在过程式程序设计语言中早就有广泛的应用，操作符重载就属于这一类多态。在 C++[3,4]中 '+' 运算符就既可用于整型变量相加，又可以用于浮点型变量相加。

(2)包容多态(Inclusion Polymorphism)。这种多态实际上建立在类型的一个子类型关系偏序集的基础之上，如果类型 A 是类型 B 的一个子类型，那么凡是类型 B 出现的地方都可以用类型 A 来替代。很明显，常规面向对象程序设计语言中由继承产生的多态都属于这类范畴。

(3)参数化多态，又称类属多态。即将类型作为函数或类的参数。例如，使用一个函数 Swap(<T>,<T>)来描述 Swap(int, int)、Swap(char, char)、Swap(float,float)等等一组函数，实际上也就给出了这一组函数的一个通用模板。特别地，在面向对象语言中，可以将类型作为类的参数，由此产生的带参数的类称为类属类。类属分两种：一是静态类属，类型参数必须在编译时刻确定。如 Eiffel 语言[5]的类属。二是动态类属，类型参数可以在运行时刻才加以确定。这一机制大大地增加了参数化多态的灵活性，但只有很少的面向对象语言给予支持，而我们目前正在参与研制的 Transframe 语言[6]提供了这一机制。

下面，我们来分析上述几种多态和动态特性之间的关系。

(1)一名多用被广泛地应用于传统过程式程序设计语言中，但是，一名多用的同一个名完全可以在编译时确定实际上究竟是哪一个名。例如，在 C++中的 '+' 操作符完全可以在编译时刻根据其操作数的类型来决定它到底是哪个版本的 '+' 操作符。一名多用只是一种语法缩写方式，并不涉及到动态特性。

(2)包容多态的一个直接结果就是对类的成员函数的动态链接。一个变量 a 虽然在程序文本中说明为类 A 的对象，但是它在程序的运行过程中可以是类 A 的任何一个子类的对象。这样，对于任意一个 a 的函数调用 f，在编译时刻无法确定 f 是类 A 的哪一个子类的函数 f。Eiffel 中经重定义的特征、C++中的虚函数都需要在运行时刻动态链接。

现在我们来看在 C++中是如何用编译手段来实现虚函数调用的。在编译时刻，编译器将每一个类的虚函数地址按一定的次序放到一张表中，该表称为这个类的虚函数表。每一个对象都具有一段存储区域，包含两部分内容：成员变量和虚函数表指针，其中虚函数表指针指向该运行时刻对象所属的类的虚函数表。编译器将对象的虚函数调用翻译成该对象虚函数表指针指向的虚函数表的间接寻址地址。

程序运行中，对象的虚函数表指针随着对象所属的类的改变而改变。例如，当说明为父类 A 的对象 a 被赋值为 A 的子类 B 的一个对象 b 时，a 的虚函数表指针被 b 的虚函数表指针覆盖。这样，当程序中出现对象的虚函数调用时，总能正确地调用到该对象运行时刻所属的类的相应函数。

虚函数表的方法实际上就是以一定的时间和空间代价来获得包容多态所带来的动态链接特性。时间代价指的是，虚函数调用时要通过虚函数表指针间址寻址一次。空间代价指的是，在运行时刻，每一个对象都要在原有数据的基础之上，增加一个指针的存储空间。

(3)静态类属本质上与一名多用无异。由于类型参数在编译时刻必须确定，它同样不涉及到动态特性。

(4)动态类属允许在类属类的类体中使用类型参数，而且类型参数可以在程序运行时刻动态地确定，并由此产生出动态时刻的新类。由于这种类在编译时刻无法具体地体现出来，只是在程序运行时刻具有生命周期，所以要处理这种动态生成的类，需要在编译时刻作大量的工作。

对包容多态的虚函数表的实现技术稍作分析不难看出，实现动态链接的关键在于：(1)每一个对象

中附加了一些信息。(2)每一个类维护了一些信息。(3)虚函数调用的间接寻址。同样,在实现动态类属类时,可以采用类似的方法。我们将凡是与类参数相关的类信息(也就是编译时刻不能确定的信息)置于该类的一张表中,称之为 Classcode 表;每一个由动态类属类实例化得到的对象具有一段 Objectcode 记录对象的一些与类参数相关的对象信息,以及一个指向 Classcode 的指针。在运行时刻对类属模板类赋予实在的参数生成新类时,需要以该类的 Classcode 为模板,填入实际的类型参数,复制出一个新类的 Classcode。而后根据已经确定的 Classcode 来确定 Objectcode 中的实际内容。这样就完成了动态类的生成及其对象的实例化。

动态类属是一种介于静态类型化和强有力的动态类型化之间的一种动态特性,为动态生成类提供了一个预先确定的模板,所以可以在编译时刻进行大量的类型检查,以保证其类型安全性,它非常适用于需要进行动态配置、具有即插即用的体系结构的系统。但是,这些优点是以大量的复杂的编译实现工作和运行时刻的时间空间代价取得的。我们不难看到,虽然采用和虚表类似的方法,但其实现复杂度已经大大地超过虚表了。

2.2 Smalltalk-80:无类型与动态特性

Smalltalk 语言[7]是由 Alan Kay 在 Xerox PARC 主持开发出的一个面向对象程序设计语言。Smalltalk 语言和建立在该语言基础之上的面向对象的图形化软件开发环境一直被奉为面向对象语言和动态开发环境的典范。

Smalltalk 的一个很大特点在于它是一个没有类型的语言,语言中的类并不是类型。对任何变量,无需对它进行类型说明,可以在运行时刻赋予它任何类的实例。这样,Smalltalk 的动态特性在包容多态的基础之上,又大大地向前走了一步。它不光是有继承关系的那些类的对象需要动态确定实际所属的类,而且所有的变量都需要在运行时刻才能知道它究竟为哪一个类的对象。

Smalltalk 利用解释手段来支持无类型这一动态特性,任何变量只有在实际解释到它时才确定它是哪一个类的对象。很明显,解释这一执行方式使得很多动态特性得以很方便自然地实现,因为解释执行手段可以有效地将那些用编译无法确定的信息推迟到运行时刻。

能否用编译的执行方式来实现无类型这一动态特性?回答是肯定的。其实,我们照样可以用类似实现动态类属的方法来做。但是,这时相应的 Classcode 和 Objectcode 会相当复杂,而且在实际运行时刻要作多得多的工作。可以想象得到,这几乎就相当于解释执行了,其效率不会得到根本改善,并且编译时刻也根本无法进行类型检查,失去了编译执行的优点。

由于不受类型的限制,Smalltalk 语言有很大的灵活性,对旨在进行原型开发的动态程序设计很方便。但是,由此引起的系统类型安全性和效率问题却大大地限制了 Smalltalk 作为一个开发最终软件产品的语言。

2.3 Self 语言:动态继承

Self 语言[8]是由 Stanford 大学的 David Ungar 和 Randall Smith 开发的一个面向对象程序设计语言。该语言是一种基于原型的语言,类和实例没有明显的区别。每个对象都是一个原型(prototype),能够作为一个模型,用以产生新的对象。由一个原型产生出一个新对象的操作称为派生(clone)。在一个原型中,状态和行为没有区别,都被称为 slot。一个原型就是一组 slot 的名及其名的列表。

派生是一个原型在其当前状态下的一个精确复制,并且派生出的原型和作为模型的原型没有任何联系。那么,各个原型对象之间是如何共享一些行为的呢?我们知道,在常规面向对象语言之中,存在着两种方式的共享:(1)利用类的实例化产生对象,所有该类的对象都具有在该类中阐述的行为。(2)通过类间的继承,子类具有父类的一些描述。

Self 中是利用代理(delegation)这一机制来使对象共享信息。每一个对象中都有一个叫做 parent 的 slot,指向其父对象。当这个对象接受到一个其无法解释的消息时,就将此消息转交给其父对象去处理(事实上,我们在编译实现常规面向对象语言的方法调用时,也是用类似的方法去做的)。这样,由于多个对象可以使用同一个父对象代理,并且父对象之间也可以代理,达到了与类实例化和继承同样的共享信息的效果。

在上述独特的语言机制基础之上,Self 语言引入了动态继承这一动态特性。由于在代理机制中,对象的 parent 也是对象的一个 slot,所以其指向可以在该对象被创建之后修改,也就是说,是动态可改变的。这就是所谓的动态继承。因此,一个对象可以在运行时刻改变它的父对象,从而改变其可以接受的消息,也就是改变它的行为。其实,从常规面向对象语言的角度而言,就是动态地改变了它的类型。

有很多实际应用都要用到动态继承。最常见的是，在一个对象被生成的时候，其所属的类不能完整地确定下来，或者在对象的生成周期中需要改动。动态继承使得随着运行时刻获得越来越多的信息，类的定义就越来越详细。例如，在一个图形系统之中，通过对现有图形对象的求交集和并集的操作，可以在运行时刻产生新的对象。但是，由于这种运行时刻产生的对象是不确定的，可以是一个矩形，也可以是一个不规则的多边形。如果是矩形，就可以采用比不规则的多边形效率更高的方法来实现一系列操作。这时，就通过动态继承的方法，根据运行时刻生成对象是否是矩形，来改变它的 parent，从而改变对它的具体操作。

和 Smalltalk 一样，Self 同样是一个没有类型的语言。由于动态继承的存在，应该说，它是一个比 Smalltalk 更加灵活的语言。同样，Self 用解释执行的手段来实现其动态继承这一动态特性。当然，我们仍然可以用编译方式来实现这一语言。显然，其实现会比 Smalltalk 语言更为复杂。

2.4 更灵活的动态机制

能否有比 Self 语言的动态继承更强的动态特性？我们可以设想这样一种语言或环境，它允许在程序运行时对一个类增加、修改或删除一个成员变量或操作，或更彻底一点，允许在运行时刻完全可以按用户的意愿来生成一个新的类。

实际上，在动态程序设计环境中，已经从环境的角度支持这样一种强动态特性。这种语言非常有利于原型速成的动态程序设计。我们知道，在采用动态运行环境开发系统原型时，常常需要在程序的运行过程中对一个类进行动态修改，如增加、修改或删除一个操作，然后继续运行系统。但是，以前对这样的一种动态机制支持都是通过环境来解决的，如 Smalltalk 环境[9]。一个重要的问题是，当一个类经过修改后它可能变成一个与原来类不相容的新类，如果直接在原来的类上进行修改，则会导致按原来类生成的对象无类可依的现象。为此，我们在正在进行的合作研究项目面向对象开发环境 MagicFrame 中引进了派生类的概念来解决此问题。当一个类经过动态修改后，我们并不对原来的类直接修改，而是生成原来类的派生类。而后相应类的对象的生成均按派生类进行；在环境中维护类与其派生类的关系。在一次开发活动结束后，开发者在环境的支持下对类与派生类的差异进行分析，然后生成一个或多个新类。

更进一步，能否直接在语言一级上提供类的动态生成与修改机制呢？从应用的角度，我们可以看到，这种语言是有其应用背景的。例如，在 Internet 上，一个远程的 Server 上有很多类型的信息，第一类信息包括一种类型的数据文件和可以对这种数据文件进行的操作，如显示，修改，查询等。这些数据文件及其操作都是该 Server 自定义的，没有标准可言。一个网络程序需要从该 Server 下载某一种信息并进行处理，但在程序运行之前无法确切地知道从该 Server 上取来的是那一种类型的信息，因而也无法确定类。只有在运行时刻，根据从该 Server 上取来的描述信息（包括数据文件描述及其上操作的描述）来实时地创建一个新的类，而后生成此类的多个对象，进行处理。很明显，这种类的的确确是运行时刻创建的，有其生命周期。一旦停止对这些信息对象的处理（如可视化显示等），且不再从该 Server 上取相同类型的信息，这些新创建的类就可以消亡了。又如，在异构网络环境下，一个程序可以根据运行时刻的不同的宿主机环境来对预先定义的类进行适当的裁剪（例如调整一些成员变量的存贮空间，由 16 位变为 32 位等）以适应网络上多种运行环境。

从实现的角度，这种非常灵活的动态特性是完全可以用解释方式来实现的。由于在运行时刻解释器可以完全控制类的全部内容，所以对类的动态生成与修改是完全可行的。但是，这样的一种语言，由于类可以动态生成和修改，其类型的安全性存在问题。如前所述，当修改一个已有实例对象存在的类时，会出现该类的原有那些对象不合法的情形，如何在语言级上解决此不一致性是值得进一步研究的问题。

3. 总结

由上述分析我们不难看出，从 C++、Transframe、Smalltalk、Self 直到上一节设想的可以动态修改类的语言，其动态特性呈越来越强的趋势。很明显，动态特性使得程序设计语言描述能力更强，使用起来更灵活，更容易处理运行时刻所带来的问题。但是，动态特性也有其不利的一面，语言实现技术复杂，运行效率降低，类型安全性不够。用解释方式来实现语言，在解释过程中需要进行运行时刻检查，其效率显然受到影响。用编译方式来实现，在空间上，需要附加一些辅助信息；在时间上，需要运行时刻进

计算机科学 1997Vol.24№.5

ODP 系统中的联邦交易模型*)

Federated Trading Model in ODP System

李 贵 尹朝万 郑怀远

(中科院沈阳自动化所 沈阳 110015) (东北大学计算机系 沈阳 110003)

摘 要 In this paper we present a federated trading model for the trading services of ODP. A federation constract is used to document the agreement between two federating traders. The federation constract consists of import constract and export constract. We discuss the constituent of the model based on the five viewpoints of ODP.

关键词 Open Distributed Processing, Federated contract, Federating trading

1 引 言

交易员是ODP系统中的一个重要组件,用于分布计算环境中对象服务的发布与获取[1],实现对象服务的分布透明与互操作,即一个对象可将其所提供的服务通过交易员进行发布,同时通过它可以获取其所需要的对象服务。联邦交易实际上是一些异构、自治和分布成员之间的一种合作机制,可以实现对象服务在整个分布计算环境中的分布功能,同时,所有用户可用以获得整个分布环境中的对象服务。

在ODP系统中,很难将整个分布环境中的所有对象服务进行集中管理,一种可行的途径是将系统中所有对象服务的管理功能分布到具有自治功能的局部交易员身上,然后在各局部交易员自治管理的基础上实现整个系统的联邦管理[2]。

在联邦交易模型中必须有一种机制来保证各局部自治成员之间的相互协作。ODP的目标是实现分布计算环境中应用与服务的分布透明性,互操作性

行一些辅助操作。

是否需要这些动态特性取决于特定应用的需要。一般说来,C++的动态链接和Transframe的动态类属基本上可以满足一般软件系统的需要,而且对程序执行效率和类型安全性影响不大,完全可以作为实现最终软件产品的语言。相反,Smalltalk、Self乃至动态性更强的语言,往往只用于对语言动态性要求较高的原型速成,出于其执行效率和类型安全性问题,很少用于实现最终的软件系统。当然,如果在计算机的体系结构方面对面向对象语言的动态特性提供支持,则可望既能发挥动态特性的优点,又可具有较高的执行效率。

进一步的工作包括:(1)对描述软件系统提出更为有效的分层动态机制;(2)探求各种各样的动态机制的高效与安全的实现机制。

参考文献

[1] Michel Beaudouin-Lafon. Object-Oriented Languages: Basic principles and programming techniques. Chapman & Hall, 1994

[2] B. Stroutrup. The C++ Programming Language. Addison Wesley, Menlo Park(CA). 1991

[3] Tasvi Bar-David. Object-Oriented Design for C++. Prentice Hall, 1993

[4] L. Cardelli & P. Wegner. On Understanding Types, Data Abstraction, and Polymorphism. Computing Surverys. 17(4)1985

[5] B. Meyer. Eiffel: The Language. Prentice Hall, 1992

[6] David Shang. Transframe: The Annotated Reference. Motorola, 1996

[7] Adele Goldberg & David Robson. Smalltalk-80: The Language and its Implementation. Addison-Wesley Publishing Company, 1983

[8] D. Ungar & R. B. Smith. Self: the Power of Simplicity. OOPSLA, 1987

[9] Adele Goldberg. SmallTalk-80: The Interactive Programming Environment. Addison-Wesley, 1983

[10] 徐家福等.对象式程序设计语言.南京大学出版社.1992

*)本项研究得到863/CIMS主题资助