程序设计 面向对象 代理机制 (3)

15-19

计算机科学Vol.20№.1 1993

代理机制及其在并发程序中的应用

蔡希尧 王小民 (西安电子科技大学软件工程研究所 710071西安)

TP311.11

摘要

After analyzing some advantages and disadvantages of inheritance in O-O languages, this paper points out that delegation (i. e., from specific to general) corresponding to inheritance (i. e., from general to specific) is also a shared mechanism of O-O languages (i. e. shared between objects, not between class and subclass). The concept and principle of delegation, and relationship between it and inheritance are discussed. Finally, some applications of delegation mechanism are outlined

在讨论面向对象程序设计的时候，我们知道对象有很高的抽象程度，模块化的特点明显，便于显式地构造并行，适宜于在并行系统设计中的应用。

人们在讨论面向对象程序设计时，还特别强调继承性的使用，把一个语言有没有继承机制，做为是否为面向对象程序设计语言的标志。一个语言使用对象但没有继承机制，称之为基于对象的语言而不是面向对象的语言，为此，Cardelli和Wegner给出了如下的公式[1]：

面向对象＝数据抽象＋对象类型＋类的继承*)

确实如此，继承性是面向对象程序设计语言的特征，这是其他语言所没有的，理应加以强调。但是，在利用面向对象的语言设计并行系统，编写并发程序时，在有效地利用对象的特点以外，能否充分地对继承性加以利用呢？回答并不是肯定的。

一、继承机制的优点和不足

在程序设计语言中引入继承机制，是一种突破，开辟了程序设计一个新的领域。语言中引入继承机制所带来的主要好处有：

1．提供了规范的等级结构，可以用树（单继承）或格（多重继承）等成熟的数学工具严格地加以描述。

2．通过类的继承关系，使公共的特性能够共享，提高了软件的重用性。

3．共同的特性可以首先加以设计和验证，然后自顶而下来开发子类，逐步加入新内容，符合逐步细化的原则。

4．通过继承性便于实现多态。包含多态是按类的包含关系来定义的；子类继承父类时，参数的类型可以不同，实现了"参数多态"；同一操作，在子类中可以重新定义，同父类中的这个操作有不同的语义，如同过载多态。

列举的这些优点，显然是非常有用的。但继承机制也带来一些麻烦的问题，主要有：

1．继承和类的密封性之间矛盾。做为一个抽象数据类型(ADT)，一个主要的特点就

蔡希尧：教授，博士导师，中国电子学会常务理事，目前从事信息系统工程、面向对象系统研究工作。王小民：博士研究生，从事分布式处理、操作系统、面向对象的分布式系统研究工作。

*) 原文为类型的继承，作者认为写成类的继承更加确切。

是密封性。实体的内部变量对外是隐蔽的，但继承性要求子类能访问父类的变量，这就与密封性发生了矛盾。对象的类是通过说明对象的实例变量来描述结构的，子类的实例必须保持父类所描述的实例变量的类型，为保持密封性。在实现继承时，是否可以直接访问父类中实例变量成为一个有争议的问题。为了达到结构继承的目的，又不致于因破坏密封性而引起不必要的麻烦，不同的面向对象的语言为此采取了不同的策略。

2．继承必须服从向上兼容的原则，子类要继承父类所持有的全部内容，不需要的也必须全部予以继承。

3．父类是在子类以前生成，不能在现有的类中生成一个描述它们共同属性的父类。

4．一个对象只能属于固定的类，限制了对象行为的可变性。

5．实现继承的技术要求父类代码的直接共享，并要求父类的可见性，包括父类的先期存在，其语法和语义表示能被子类的编译或解释系统所访问，这些要求在多机环境下的实现是很困难的。

6．子类继承父类时，可以加入新的特征（属性和操作），因此，经过继承所得到的对象，其行为将不同于父类的对象。在这些新的对象中，所有继承下来的老的特征的类型仍然保持不变，因此对于父类的实例是合法的任何场合，子类的实例都是可以应用的。这样，将导致语义上的模糊。例如，人们比较自然地会要求与父类实例有关的某些不变量（invariants）在子类的实例中完整地保存下来，但这一点不能用类型化机制加以检验；而对系统的行为进行推理时，上述的要求还不足以认为新的对象是老的类之中的一个成员。

当借助于继承定义一个新类时，需要一个新的体，否则，新加入的特征将无法加以利用。但是，一个新的不同的体，意味着它的动态行为可能与老的完全不同。因此，如果沿着这样的一条继承路径展开，以形式的方法对系统的行为进行推理以证明程序的正确性，将变得非常困难，甚至于不可能。这对于并发程序设计是个严重的问题，因为并发程序比顺序程序更需要正确性验证。

继承性与类型化的结合，还会引起其他的问题。为了最充分地利用继承性，必须允许在子类中对操作的再定义，使得在响应消息以后，操作的执行依赖于接收消息的对象实际上是属于哪个类（父类或子类），于是要问：在充分利用继承机制时，要不要类型化机制？要的话，对于父类和子类中的操作类型需要加以什么样的约束？

无可否认，在语言中引入继承机制是一大进步，面向对象语言比传统语言更具有吸引力的主要原因之一也在于引入继承性，使语言更有活力，能适应广泛的应用要求。但是，刚才指出的继承性所带来的问题，也是不可忽视的，特别对于并发程序设计，尤其严重。针对这些问题，一方面要进行理论上的探讨，以深入理解继承的实质；另一方面，须找寻其他的共享机制，这正是我们要介绍代理机制的目的。

二、代理机制

和继承相似，代理也是面向对象程序设计语言中的共享机制。继承是建立在集合的概念之上，继承关系意味着集的包含关系。建立继承首先要找出同类事物中所有成员的共同特点构成类，以供共享；每个对象是类的一个实例。代理则是建立在"原型"的概念之上，其特点是在同类事物中找出一个对象做为原型，当其他对象和原型有相同的内容时，则不必自行设置，可以由原型代理。这两种共享机制在概念上的不同，用通俗一点的话来说，继承是从一般到个别，从父类派生出子类；代理则是从个别到一般，后来的对象可以与原型作比较，发现有共同之处则实现代理。支持用代理实现共享的人们所持基本理由就是：从个别到一般更符合人们的认识过程，而从一般到个别所以能够做到，

首先要对许多个别的对象进行观察，认识它们的共性，才能抽取出一般，形成更高的抽象。

用Lieberman所设计的笔和乌龟的例子[2]，能很好地说明代理机制的原理。笔的特点是能够在平面上写字或作画，以(x, y)表示平面上的位置，对笔的操作用 Draw 表示，则笔这个对象可以写成

(x, y, Draw)

其中(x, y)是对象笔的变量，Draw是它的操作。

乌龟和笔是不同类的事物，但乌龟在地面上爬行，和笔在纸上写字是相似的，可以用(x, y)表示乌龟的位置，Draw用来表示爬行动作，这些在笔这个对象中都已经加以定义。但对乌龟来说，还有一些其他的特点，乌龟的头所对的方向，用Heading表示；笔可以前后左右自由地运动，而乌龟主要是向前爬，于是可以加上Forward这一特点，于是对象乌龟表为

(x, y, Draw; Heading, Forward)

其中，(x, y, Draw)可以用笔来代理，笔成了乌龟的原型。

这个例子中笔和乌龟两个对象，如果用类和继承性来描述时，可能的途径是：先组成一个类，包含笔、乌龟以及和它们相近的其他对象的共同特征，一支笔和一个乌龟，则通过对这个类的继承而得到。

一个代理的原型系统，有以下一些特点：

1．没有类和实例的区别，任何一个对象都可以作为原型。

2．实例可以互相共享，而不是彼此独立。

3．实例的生成只涉及当前还不能共享其他实例的那些属性结构与行为，和描述共享的消息传递，而不是沿着继承路径展开。

4．实例可以改变与原型的从属关系，这在继承机制中是不允许的。

5．在继承机制中，一个对象可以通过生成它的类或更高的父类继承信息，但控制权是归自己的。在代理机制中，控制则交给代理的对象。

从这些特点可以看出，代理比继承较为灵活，它可以节省存贮空间，但运行速度比不上继承机制。代理机制的灵活性影响了它所塑造模型的语义明确性，在语义的认识和执行上都比较困难，这是缺点。

在原型系统中，当一个对象收到一个消息时，它首先用存贮于它的专用部分的行为加以响应，如果不能应答消息中指明的请求，则将消息送给一张包括若干原型的表，找寻能够响应请求的原型，如果找到了，则由这个原型代理。

代理机制的出发点虽然与继承机制不同，但两者都是面向对象语言中的共享机制，从这一共同点来看，存在着一致性。所不同的是，代理机制的共享是在对象和对象之间，而继承机制的共享是在类和子类之间，但实现共享时都使用消息传递来达到彼此间的交互作用，这也是一致的，况且类可看成是一个对象。Stein根据两者的共同点，指出代理和继承是可以互为模型，互相转换的[3]。在实际的应用中，强调的是两者的区别，但必须看到它们各自的优缺点。对于继承和代理机制的讨论，使人们得到的一个共识是：面向对象的设计空间是宽谱的，有两个机制，即新对象的生成和对象间行为的共享是基本的，对这两个机制所能提供的灵活程度，形成了不同的语言特色，代理和继承则处在谱的两端，在这中间，还可以有其他特色的面向对象的语言，近年来提出来的基于规则的代理[4]和可控的代理[5]等，体现了这方面的努力。

三、代理的应用

代理机制已在近年来新开发的语言中得到应用，通过两个例子来说明应用情况。

1、演员(actor)语言中的代理机制

演员是一种面向对象的计算模型，适用于并行计算，是麻省理工学院的Carl Hew-

itt及其同事们所并发的[8]。

演员是一个实体，是主动的对象，它和消息是构成演员系统的两个基本元素。每个演员有一个信箱地址和一个行为。演员之间的通信方式是消息传递。发送消息的演员必须知道接收者的信箱地址。演员也可以向自己发送消息。当多个演员向同一演员发送消息时，为保证每次通信的完整性，要设法使多个输入消息串行化。信箱的地址可以自由地在系统中传送。这给予系统以重构能力，使演员系统的互连是动态可变的。外部的信箱地址可以送入系统，因此，演员系统是可扩展的。

一个演员接收到另一演员发来的消息，要予以响应。演员在每一时刻所具有的特定行为，决定了接收到消息后应做出什么响应。一个行为一般包括三个动作：（1）向知道信箱地址的其他演员发送消息，或向自己发送消息。（2）创建一个新演员。（3）对替换(replacement)的计算。替换是实现局部状态的改变，以便处理下一次接收到的消息，但保持程序中所用的标识的引用透明。替换计算可以在上一次消息中的其他动作仍在执行时进行，因而具有内在的并发性。

演员之间的通信是异步的，一个演员向另一个演员发出消息后，不必等待回答，可以立即去处理其他的消息。从整个系统来说，所有的东西都当做演员，包括函数、子程序、进程、数、表、文件、设备等，它们都有能力接收消息，而所有演员对动作的执行是并发的。所以一个演员系统，有高度的并行性。

按照演员系统的特点，已经设计了多种语言。面向对象语言的所有特点，在演员语言中都可以采用，但强调的方面有所不同。代理机制所具有的灵活性和所依据的概念，适合于在演员系统中的应用。以语言Act1为例[7]，当一个演员接收到一个消息，而它自己所持有的知识不能够给予响应时，它将发送消息请其他演员代理。在Act1中，发送消息给代理演员的演员，叫做顾主。多个顾主可以共享相同的代理演员。在设计程序时，把多个演员经常要用到的共同知识，集中在一个演员中，使用时由这个演员代理，这样，避免了在每个演员中重复设置共同的知识。

在Act1中，有两类演员，一类叫底层演员，它们可以使用实现语言Lisp的基本数据和函数；另一类叫脚本演员，它们有显式存贮的脚本和代理者，Act1的解释程序的基本循环是：当一个目标演员接到消息时，如果它是底层演员，在底层脚本中的表中找出演员类型，启动脚本；如果目标演员是脚本类的，抽出脚本并启动它；如果脚本不接受消息，则将消息转送给代理演员，代理者成为新的目标演员。

2、并发语言中的代理机制

PROCOL是有代表性的面向对象的并发程序设计语言[3]，采用代理机制。语言中的多个对象可以同时执行。同一时间内，每个对象只执行一个动作，但允许特殊的中断动作中断正常的动作。通信是以远方过程调用(RPC)的方式或短时联编的单路同步的方式进行。用显式的协议控制对象的访问，协议不是一个对象的可执行部分，其功能如同一个说明性的规范，以表明服务员和它的顾客之间的合法的通信，并提供顾客的类型检验。通信联编是动态的，适用于分布的、增量式的和动态的对象环境。

在PROCOL中使用代理机制有两个理由：第一是有较好的信息隐蔽；第二是在动态的对象环境中，对象之间的接触，以及由此形成的共享，是短时的。继承是静态的，继承关系一旦确定就保持不变。代理更象动态连接和拆除，适宜于动态的对象环境。

PROCOL允许动作的全部或部分代理。动作可以分布在若干个代理者之上。所有分布式的代理可以并行地运行。代理的原语为

@ Delegate Object. Action Name msg

其中msg并不一定和代理动作所接收的消息

相同。应用这一原语的实际例子可参考[8]。

如同Act 1采用Lisp为实现语言一样，PROCOL采用C为实现语言。

四、一个混合模型

代理机制的灵活性是一个突出优点，而继承机制的等级结构对许多实际应用都是相当可贵的，如果能够把两者结合起来，利用它们的优点，采取措施克服或减弱它们的缺点，这样的系统将是可取的，对于一些特定的应用，这也是容易做到的。我们正在作这方面的尝试。需要解决的主要问题是：利用原型来模拟类和继承，给予对象的结构和行为以及消息传递以一定的约束。在这样的系统中，对象或者是类，或者是实例，而对象的密封程度将有所不同。

在代理机制中，一个对象要求另一个对象（原型）代理，使用了原型中的代码，和一个子类享用父类中的代码是相似的，而通信的方式也是相似的，对象和原型之间，子类和类之间，都是用消息传递实现交互作用。这样，我们有可能把若干个对象的共同部分（属性和方法）集中在一起构成一个对象，这个对象具有类的特点，而原有的对象如同子类的实例，而它们的共同操作，则可以通过固定的路径由相当于父类的对象代理。

对于对象的结构和行为以及消息传递的约束主要是对访问对象的控制，以及对象之间的关系，新的对象可以在什么条件下生成，包括什么内容等等。这些约束实际是一些规则或条件，它们必须显式地加以说明，以满足应用的需求，便于用户加以设置或修改。当一个对象收到代理的消息时，要计算这些约束条件，以决定如何响应。

系统的设计工作主要在于对象以及它们的特性概括（以形成共享的对象），消息传递规程，约束条件，以及有关的语法和语义的描述。代理虽然是很有用的机制，但没有专用的语言，新的系统的实现只能借助于现有的语言，加以必要的扩充。

参考文献

[1] Cardelli, L. and Wegner, P., On Understanding Types, Data Abstraction, and Polymorphism, ACM Computing Surveys, Vol. 17, No. 4, Dec. 1985

[2] Lieberman, H., Using Prototypical Objects to Implement Shared Behavior in Object-Oriented Systems, ACM SIGPLAN Notices, Vol. 21, No. 11, Nov. 1986

[3] Stein, L. A., Delegation is Inheritance, ACM SIGPLAN Notices, Vol. 22, No. 12, Dec. 1987

[4] Almarode, J., Rule-Based Delegation for Prototypes, OOPSLA'89 Proceedings, Oct. 1989

[5] Minsky, N. H. and Rozenshtein, D., Controllable Delegation, An Exercise in Law-Governed Systems, OOPSLA'89 Proceedings, Oct. 1989

[6] Agha, G. A., ACTORS, A Model of Concurrent Computation in Distributed Systems, The MIT Press, 1986

[7] Lieberman, H., Concurrent Object-Oriented Programing in Act 1, in «Object-Oriented Concurrent Programming», ed by A. Yonezawb and M. Tokoro, The MIT Press, 1987

[8] Bos, Jan van den and Laffra, C., PROCOL, A Concurrent Object-Oriented Language with Protocols Delegation and Constraints, Acta Informatics, Vol. 28, No. 6, July 1991