基于 ACTOR 模型的并发面向对象语言 AC++*

董 哲 刘 琳 田籁声

(吉林大学计算机科学系 长春 130023)

摘要 AC++是用 ACTOR 模型建造的并发 C++语言. 本文介绍 AC++的设计与实现,着重探讨如何在语义级上平滑地结合 ACTOR 模型和普通面向对象语言,提出了"扩充的行为抽象"和"异步创建"等新方法,使新语言既能保持面向对象语言的特性,又能支持 ACTOR 模型提供的描述并发计算的能力.

关键词 并发面向对象语言,ACTOR 模型,扩充的行为抽象,异步创建,运行支撑系统.

面向对象程序设计方法是当今最有前途的软件技术之一. 面向对象方法是与现实的模型相对应的,而现实模型中的对象是并发活动的,因此面向对象方法被认为具有潜在的并行性. 但是普通面向对象语言如 C++等由于其延续传统的过程式语言的单指令流、单数据流模型,都缺乏对这种固有的并发性的支持.

ACTOR 模型最早由 Hewitt 于 1977 年提出,经过许多人的扩展,后来 Agha[1] 总结定义了一个由少数表达能力强的原语组成的 ACTOR 模型. 由于其提供的封装和消息传递等机制与面向对象方法非常相似,而且 ACTOR 模型又具有灵活的控制结构和强大的并发描述能力,因此成为并发面向对象语言的重要的基础模型之一.[2] 近年来,许多人以 ACTOR 模型作为基础,开发出新的并发面向对象语言,如 ABCL/1, ACT++[3], ALBA 等.

本文主要探讨如何简单地在语义级上平滑地结合 ACTOR 模型和普通面向对象语言,构成新的并发面向对象语言. 我们以 C++为基础语言,构造的新语言称为 AC++. 在 AC++中采用了"扩充的行为抽象"和"异步创建"等新方法,使得 AC++既支持 ACTOR 模型的并发计算能力,又能保持面向对象语言的特性.

1 ACTOR 模型

ACTOR 模型是一个并发计算模型,计算是通过 actor 间的消息传递来完成的,actor 是

* 本文研究得到国家自然科学基金资助. 作者董哲,1971 生,博士生,主要研究领域为计算机网络与分布式系统. 刘琳,女,1973 年生,博士生,主要研究领域为计算机网络与分布式系统. 田籁声,女,1938 年生,教授,博士导师,主要研究领域为计算机网络与分布式系统.

本文通讯联系人:董哲,长春 130023,吉林大学计算机科学系

本文 1996-03-14 收到修改稿

并行计算的最小单位. ACTOR 模型中有 5 个基本元素:actors、邮箱、消息、行为和熟人.

一个 actor 是一个自含的、交互的、相互独立的活动对象. actor 间的交互只能通过异步消息传递发生. 每 1 个 actor 包含 1 个邮箱,其地址是 actor 的唯一标志,传递给该 actor 的消息放入邮箱中缓冲. 存储在邮箱中的消息按先到先服务的顺序被响应. 一个 actor 只能通过向别的 actor 发消息来改变另一个 actor 的行为,而且一个 actor A 能够向 actor B 发消息,当且仅当 A 知道 B 的邮箱地址,B 也称为 A 的熟人. 邮箱地址本身可以作为消息的一部分传送,从而 actor 间的连接可动态改变. actor 的行为说明 actor 如何响应一个请求消息. 行为是通过一段称为"行为描述"的代码体来定义的,包含一组可以响应的方法定义和一个熟人表. 每个 actor 在它的生存期的任何时候都是处在一定的行为之下的,按照该行为响应其它 actor 发来的方法请求消息.

在处理 1 个消息时,1 个 actor 可以做 3 种操作:生成新 actor、发消息给熟人和指定一个替代行为. 在 Agha 的模型中,分别由 3 个原语表示:new,send 和 become.

new 原语用来生成新的 actor,返回值是所生成的 actor 的邮箱地址. 这使得 actor 可以在计算中按需要动态生成. 新生成的 actor 的邮箱地址可以被传给其它的 actor,其它 actor 就可以向新 actor 发消息. new 操作中要给新 actor 指定一个初始行为.

send 原语用来进行消息传递,一个消息包含目的 actor 的邮箱地址、要调用的方法名和引用该方法所需参数. 所发送的消息在目的 actor 的邮箱中被缓冲,因此 send 原语执行时不阻塞,是异步消息传递方式.

become 原语用来指明一个替代行为. 一个 actor 的行为确定了其处理一个消息时可以采取的操作. 替代行为是指 actor 在处理完本消息后所处的行为,也就是 actor 用指出的替代行为来处理下一个消息. 当指定的替代行为与当前行为不同时,actor 的行为就在运行时动态地被改变. 一个 actor 在不同的时间点上可能表现出不同的行为,这些行为可能没有相同的方法或数据结构,在 actor 的生命中唯一不变的是邮箱地址. become 操作需要指出替代行为名和熟人表.

2 AC++语言中的 ACTOR 模型

本节从 ACTOR 类的表示、行为的表示及 actor 之间的消息传递原语等方面描述在 AC++中的 ACTOR 模型.

2.1 扩充的行为抽象

在 ACTOR 模型中,最新奇的概念是行为[3],如何表示行为是表示 ACTOR 模型的关键之一. 用面向对象的观点来分析 ACTOR 模型中的行为定义,会看到它和类定义是相似的,即具有对内部信息的封装和外部可引用方法的定义. 在这种分析下,替代行为实际上就是在共享同一邮箱地址的不同类实例间的切换. 换言之,就是一个对象处理完一个消息后,可以例化其它类的一个对象,并把控制权和邮箱地址交给它,让它继续处理邮箱中的其它消息. 这使得 actor 可以动态地改变自己的行为,使 ACTOR 模型具有很高的灵活性和开放性.

并发面向对象语言的并发性要求并发对象间有同步关系,由于对象的内部状态只能由方法响应来改变,因此并发对象的并发控制一般在对象内部实现,这导致了继承下来的代码

可重用性和对象的封装能力降低.因此并发性和继承性被认为具有相互干扰的趋势.[4]

为解决这一矛盾,D. Kafura 在文献[4]中提出"行为抽象"的方法:将 ACTOR 模型的行为定义抽象为一个对象向外输出的方法的集合,不同的行为可以定义为不同的方法集合,用行为名指出不同的方法集合.一个 actor 的所有行为都被封装在一个特殊的类中,称为 ACTOR类.每个并发对象 actor 都是由某个 ACTOR 类派生出来的.每个 actor 在任何时刻都处在一定的行为下,处在不同行为下的 actor 向外输出不同的方法组,即 actor 仅响应由行为名所指定的方法组中的方法的调用请求.actor 在方法响应时可以指定自己的替代行为."行为抽象"利用行为定义把并发控制从对象内部抽取出来,使并发控制可以作为一个独立的部分被继承,排除了并发性对继承性的干扰.但由于限制了 become 操作的范围,牺牲了 ACTOR 模型的"开放性".[5]

从语言定义和实现方面考虑,我们认为"行为抽象"有如下优点:①"行为抽象"使 ACTOR 类的定义靠近普通类定义,利于 ACTOR 模型与普通面向对象语言的结合.②实现中没有 actor 间的上下文频繁切换的问题,可减少系统开销.③对 become 操作进行适当的限制,可提高程序的可读性和可维护性.④为并发类的继承问题提供了一种可行的方案.因此,我们认为行为抽象是一种较好的表示行为的方法.

在 D. Kafura 提出的"行为抽象"中,方法定义是与行为定义完全独立的,只有在行为定义中列出所能响应的方法名后,才将方法和行为联系在一起.这使得在不同的行为下,对同一个方法请求只能做出同样的响应方式.而在原 ACTOR 模型中,不同的行为下,对相同的方法请求完全可以给出不同的响应.因此,"行为抽象"在这一点上限制了 ACTOR 模型的灵活性.

我们在 AC++ 中提出了一种"扩充的行为抽象"的方法,目的是不仅在行为定义时指定该行为是否响应某个方法,而且要表示出同一方法在不同行为下可以做出不同的响应.我们在方法定义中,引入了行为域,指出该方法定义属于的行为,即在那个行为下向外输出该方法体.这样,在同一个 ACTOR 类中,可以定义具有多个方法名和参数完全相同的输出方法,仅在行为域中分别指出它们所属的行为.运行过程中,可以在不同的行为下动态地决定对外输出的方法体.我们称之为"基于行为的方法重载".在方法定义中,行为域也可以缺省,这时我们规定其用于所有未专门指定方法的行为.

下面我们给出用"扩充的行为抽象"表示的 ACTOR 类,其结构如下:

```
ACTOR actor_class  name ;super_class
    { behavior :
        behavior1 = {b1_method1(),b1  method2(),...}
            ...
        behaviorn = {bn_method1,bn.method2,...}
    private :
            ...
    public :
        return_type method1(args ...)
            ...
        return_type methodn(args ...)
    }
    return_type method1(args ...) : behavior_k1
        {...}
    ...
```

```
return_type methodn (args ...) : behavior_kn
{...}
```

在 ACTOR 类中，比普通类多出一个 behavior 定义域，定义了行为名和该行为的输出方法组. 在子类中可以对父类的行为进行继承和修改. 每个输出的方法都在公共区作说明. 每个方法定义后可以有一个行为域，指出该方法在哪个行为下用本代码体响应外界的方法调用请求.

2.2 actor

AC++语言中支持 2 类对象：主动对象和被动对象. 被动对象是 C++中由关键词 CLASS 标志的普通类的实例，没有自己的控制线索，与原 C++的对象语义完全相同. 主动对象有自己的控制线索，是由关键词 ACTOR 标志的一种特殊类的实例，能够进行异步消息发送和接收，并与其它主动对象并发执行，也就是 actor. 其中我们把 main()函数看作一个主动对象. 程序运行从 main()函数开始，一个激活的主动对象可以激活其它主动对象与本对象并行执行.

2.3 替代行为

在 actor 并发对象的存活期间中的每一时刻，都处在一定的行为之下，行为的转换是方法响应中完成的，一般来说，方法定义的最后都要由 become 原语指定替代行为. 如下

```
become(replace_behavior);
```

在未指定替代行为时，认为指定了当前行为作为替代行为. 在任意时刻，actor 总是响应当前行为所指定的可以响应的方法，而其它方法请求则被挂起.

2.4 actor 的异步创建模式

actor 可以在变量声明时例化，也可以在运行过程中动态创建. 在变量声明时例化如下

```
actor_class_name actor_1(arg1,arg2,...);
```

actor 并发对象的名字代表一个 actor 的全局唯一的邮箱地址，该地址可以在 actor 之间进行传递，知道该地址的 actor 就可以对该 actor 进行方法请求，这样可以动态改变 actor 之间的拓扑结构.

在我们见到的关于并发语言的研究工作中，其动态创建模式都是同步的. 同步创建模式是与传统的单线索程序相适应的，在并行运行环境下，同步创建模式实际上限制了并发程度. 参照异步消息传递的方式，我们提出了异步创建模式. 异步创建模式是指首先提出创建请求，不等待创建结果，程序继续向下运行. 当需要结果时，再接收创建的结果. 这种异步创建模式的优点在于可以使得多个创建请求被并行处理，也可以使创建请求和异步方法请求被并行处理，可以进一步增加并行度，以提高系统的性能.

当在运行中动态创建 actor 时，即 ACTOR 模型中的 new 原语操作，首先用如下非阻塞原语 asyn_create 提出创建请求，创建名为 actor_class_name 的并发类的一个实例，其构造函数参数为 arg1,arg2,.... 返回为本次创建请求的标识号 asyn_create_id，该标识号用于 actor 唯一标识该次创建请求. (int)asyn_create_id=asyn_create(actor_class_name,arg1,arg2,...)；程序可以继续处理其它事物，当需要创建的 actor 的地址时，用如下阻塞原语 receive 接收标识号为 asyn create id 的创建请求的处理结果，实现按需等待

```
receive(asyn_create_id,&actor_1);
```

得到的结果是被创建的actor的全局唯一的邮箱地址,其使用方法和在变量声明中产生的邮箱地址完全相同.

2.5 actor间的异步消息传递

ACTOR模型中的actor间的消息传递是异步的,因此AC++中的actor间的消息传递也采用异步方式,即调用方在发出方法调用请求后,不等待结果的返回,继续向下运行,在需要用到本次调用请求的结果时,再进行阻塞式接收.使得调用方和被调用方在最大程度上并发执行.我们提供了asyn_call,receive和reply 3个原语描述并发对象间的异步消息传递.调用方首先用非阻塞原语asyn_call发出调用请求,向名字为actor_name的actor发出名字为method_name的方法调用请求,其方法调用参数为arg1,arg2,...返回为唯一标识本次方法调用请求的标识号asyn_call_id. (int)asyn_call_id=asyn_call(actor_name,method_name,arg1,arg2,...)

提出请求后,程序可以处理其他事物,当需要本次的方法调用请求的结果时,用receive原语进行接收,得到结果返回result中.如果方法调用结果尚未返回,receive语句阻塞,直到方法处理成功并返回结果.

receive(asyn_call_id,&result)

actor在处理方法调用请求时,如需返回结果,要使用reply语句进行结果回送.如下

reply(result)

reply原语可以出现在方法定义的任何地方,仅表示结果的回送,不表示方法的结束.可以按需要回送结果,当不需要返回结果时,也可以不出现.

3 实　现

我们采用预编译方法实现了AC++语言,语言实现过程中兼顾到语言的运行支撑系统.为检验语言的性能,我们还设计了该语言的运行支撑系统.[6]如图1所示,预编译器具有双重功能.一方面对AC++中特殊的语言机制进行识别和转换,把源程序按ACTOR类编译为多个可以并发执行的C++模块;另一方面自动建立与运行支撑系统间的通用接口,为运行支撑系统的设计与实现提供方便.通用接口[7]是我们提出的并发语言和运行支撑系统间的接口标准,主要有如下特点:结构简单、清晰;提供的信息完备;具有通用性,即不依赖语言的具体运行环境.

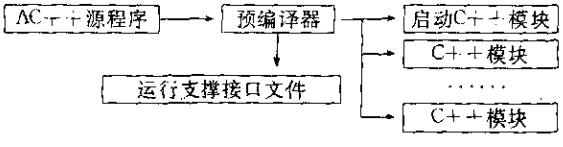

图1 预编译器处理过程

通用接口包括静态和动态2部分.静态部分包括在预编译时生成的、程序运行时需要用到的全部信息.如图1所示,预编译器除生成可并发运行的C++模块外,还产生一个运行支撑接口文件,其中包含运行支撑系统在运行时需要了解的关于AC++源程序及所产生的C++模块的全部信息,例如动态创建时需要使用的ACTOR类名到C++模块文件名的变换表等.

动态部分是指并发模块(即actor)在运行时,与运行支撑系统的交互界面.如图2所示,对应于ACTOR模型的邮箱,我们定义6个缓冲队列,作为actor与运行支撑系统的接口.

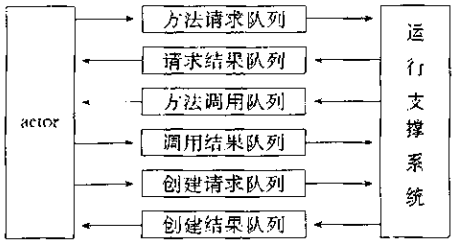

图2 actor与运行支撑系统接口

这样，运行支撑系统相应于ACTOR模型中的邮递系统. 在运行过程中actor仅通过对这6个队列的处理来与其它actor进行信息交换.

在预编译后，关于通信及创建的原语分别被转换成如下分别对以上6个队列进行操作的程序段：asyn_call原语转换成将方法请求打包送入方法请求队列尾，包结构如图3所示. 该请求包由运行支撑系统按邮递地址传送给相应的actor，进入方法调用队列中排队，服务方的actor主动从调用队列中取出该包，拆包，进行相应的方法调用；reply原语转换成将结果打包送入调用结果队列，包结构如图4所示，结果包由运行支撑系统按返回的邮递地址传送给相应的actor；receive原语转换成按请求结果的类型，分别查看请求结果队列或创建结果队列，如果发现该请求的结果包，进行接收，否则等待直到结果返回；asyn_create原语转换成将创建请求打包，如图5所示，送入创建请求队列，创建请求包由运行支撑系统进行处理. 其结果包由运行支撑系统生成，送入创建结果队列，结构如图6所示.

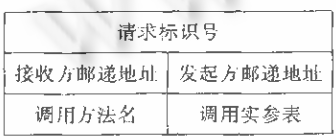

图3 方法请求包

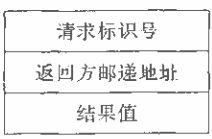

图4 请求结果包

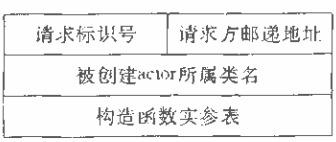

图5 创建请求包

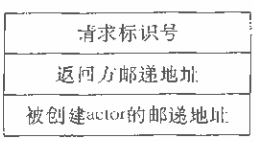

图6 创建结果包

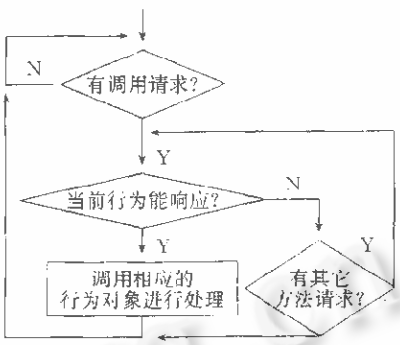

图7 actor控制线索

在预编译中，每一个ACTOR类定义被转化为一个可以独立运行的C++模块. 创建该ACTOR类的一个actor，就是运行该C++模块产生的一个进程. 这样，一个ACTOR类对应于转化出的C++模块，而它的多个实例(即actor)对应于由该C++模块运行的多个进程. ACTOR类在转化过程中，每个行为定义转化为一个普通类定义，我们称为行为类. actor的不同行为在C++模块中表示为用不同行为类的实例对外部方法调用进行响应. 行为在预编译中编号，模块本身维持一个行为状态变量，其值为行为的标识号，指出本模块当前所处的行为. 在模块的开始部分，有多少个行为类就例化多少个行为对象. 在不同的行为下，用不同的行为对象响应外部的方法请求. 行为类的方法中的become原语操作被转换为对行为状态变量的操作.

actor是主动对象，拥有自己的控制线索. 因此在ACTOR类向C++模块的转化中，我们在C++模块中加入控制线索，使得每个actor被创建后，按其自己的线索运行. 加入的运行线索主要为从方法调用队列中取方法请求包，拆包，判断是否是当前行为下可以响应，如不是，查下一个. 否则，调用相应的行为对象的相关方法进行响应. 如图7所示.

4　相关工作比较

我们的工作主要参考 ACT++,因此我们只给出与 ACT++的比较. ACT++在 C++中建立了一套类体系来描述 ACTOR 模型,并提出了"行为抽象"的概念,解决了并发性与继承性的干扰,被许多文献引用,是较典型的方法. 我们的工作与其相比,有以下不同:ACT++是基于类库方法实现 ACTOR 模型,而我们是直接对语言机制进行扩充,增加了 ACTOR 类和 5 个原语来表示 ACTOR 模型;ACT++是在单机上用 LWP 模拟实现的,我们对分布式实现提供支持;我们在"行为抽象"的基础上增加了"基于行为的方法重载",增加了模型的灵活性. 并提出了"异步创建"模式,提高了并行度.

5　结束语

本文介绍了在普通面向对象语言中实现 ACTOR 并发计算模型的尝试,AC++已经在 Sun 工作站上初步实现. 由于我们的工作是实验性的,AC++在许多方面还有待完善,还需要做大量深入细致的工作.

参考文献

1　Agha G. Actors: a model of concurrent computation in distributed systems. MIT Press, Cambridge, MA, 1986.

2　Agha G. Concurrent object-oriented programming. Commun. ACM, Sept. 1990, **33(9)**: 125～135.

3　Kafura D, Lee K. ACT++: building a concurrent C++ with actors. Journal of Object-Oriented Programming, May/June 1990. 25～37.

4　Kafura D, Lee K. Inheritance in actor based concurrent object-oriented languages. In: Proceedings of ECOOP'89, 1989. 131～145.

5　Nelson M L. Concurrency & object-oriented programming. SIGPLAN NOTICES, Oct. 1990, **26(10)**: 63～69.

6　刘琳,董哲,田籁声. 并发面向对象语言的运行支撑系统 DRTS. 吉林大学自然科学学报,1996, **3**: 5～8.

7　田籁声,黄莲淑,夏滨. 分布式程序设计语言 DC 及其在松散耦合分布式环境中的实现. 软件学报,1995, **6(7)**: 399～406.

A CONCURRENT OBJECT-ORIENTED LANGUAGE AC++ BASED ON ACTOR MODEL

DONG Zhe　LIU Lin　TIAN Laisheng

(*Department of Computer Science　Jilin University　Changchun*　130023)

Abstract　AC++ is concurrent C++ based on actor model. This paper introduces the design and implementation of AC++. The authors proposed some new skills, such as "extended behavior abstract" and "asynchronous creating", which make the new language maintain basic features of object oriented language and support parallel computing as well.

Key words　Concurrent object oriented language, ACTOR model, extented behavior abstract, asynchronous creating, run_time support system.