CORBA中POA策略研究

许丽婷,董丽丽
(西安建筑科技大学 信息与控制工程学院,陕西 西安 710055)

摘 要:可移植对象适配器(POA)是重要的CORBA组件,它使得服务器端应用程序能方便地在对象请求代理(ORB)实现中进行移植。文中对当前POA的工作原理进行了全面阐述,提出一套新的POA调度方法,该方法在访问ORB过程中,通过不同策略的优化组合,使得服务器端应用程序性能得到显著提高,应用表明了该优化策略是有效的。

关键词:CORBA;对象请求代理;可移植对象适配器;策略
中图分类号:TP311 **文献标识码**:A **文章编号**:1673-629X(2007)01-0060-03

The Research of POA's Policy in CORBA

XU Li-ting, DONG Li-li
(Coll. of Info. & Auto-Control Eng., Xi'an Univ. of Architecture & Tech., Xi'an 710055, China)

Abstract: POA is an important component of CORBA. Application program of server can be transplanted in realization of ORB by POA. Explains POA's working principle completely and brings forward a new degree method of POA. Through optimization and combination of different policies, the measure makes the performance of application program in server improved evidently. Application indicates that the optimization policy is effective.

Key words: CORBA; ORB; POA; policy

0 前 言

CORBA是由国际OMG(Object Management Group)组织推出的分布式对象计算标准,即通用对象请求代理体系结构。CORBA用于解决分布式面向对象应用中的互操作和可移植性问题的设计架构。CORBA的核心对象请求代理(Object Request Broker,ORB)提供了分布式远程对象间对象定位、激活和通信的透明性[1],它作为一个"软总线"来连接网络上的不同对象,提供对象的定位和方法调用,并负责代理客户请求与对象实现间的通信。可移植对象适配器(Portable Object Adapter,POA)是ORB的重要组成部分。POA是介于ORG核心和服务器程序之间的软件层,它是服务器端众多应用程序中的一个相对独立的实体,它标准化了接口定义语言(LDL)编译器产生的框架类,以及POA和Servant之间的交互,使得服务器应用程序能在不同的ORB实现中得到移植[2]。图1为CORBA的体系结构。

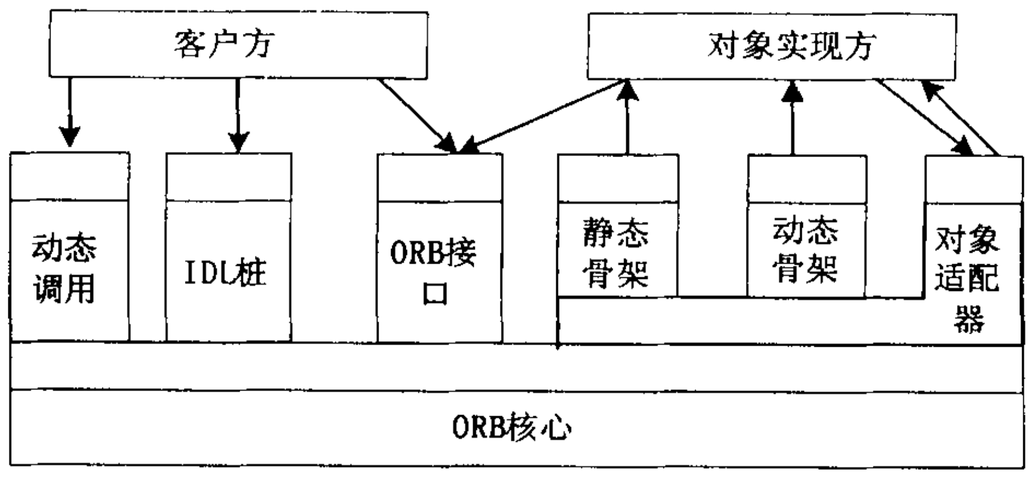

图1 CORBA的体系结构图

1 POA的工作原理

POA作为ORB的成员,仅仅对服务程序是可见的,客户程序并不知道POA的存在[3]。在服务器上,一个Servant在POA中注册以后,首先由服务器端以某种方式为CORBA对象创建一个对象引用,其中包含有目标对象的ObjectId,以及与创建目标对象引用相应的POA标识。客户端得到对象引用后,发出调用操作的请求。ORB接收到一个请求后,会异步地将请求传输给POA,即将客户端的请求发送到服务器端。一旦对请求的处理结束,POA会通知ORB将其结果返回。具体的实现过程如图2[4]所示。

(1)当客户机发出一个请求时,ORB首先要找到一个适当的服务器,然后它在这个服务器中根据POA标识来定位适当的POA。

(2)如果在服务器进程中POA不存在,ORB使用一个AdapterActivator创建所需的POA。如果没有AdapterActivator,客户机就收到一个OBJECT_NOT_EXIST异常[5]。

收稿日期:2006-04-17
基金项目:陕西省自然科学基金项目(2001X13)
作者简介:许丽婷(1974-),女,江苏徐州人,硕士研究生,研究方向为网络与分布式系统;董丽丽,副教授,研究方向为网络与分布式系统。

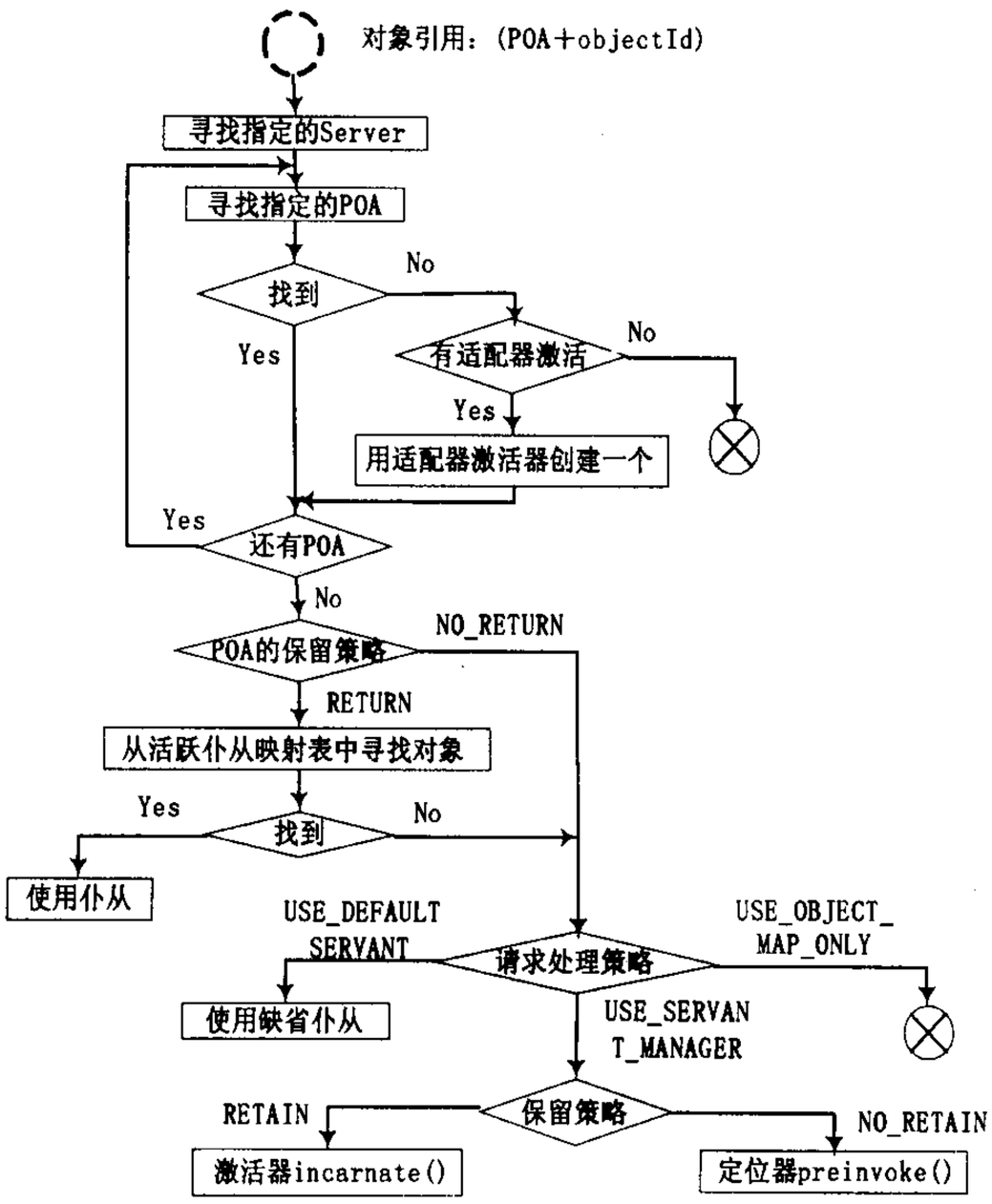

图 2　仆从的建立、使用和中止

(3)一旦 ORB 找到适当的 POA,它就会将请求传送给这个 POA。请求的进一步处理取决于此 POA 采用的策略以及对象当前的激活状态。

如果 POA 的 Servant 保持策略取值为 RETAIN,POA 就会在活跃对象映射中查找,看是否有一个 Servant 与请求中的 ObjectId 值相关联。如果有,POA 就会从活跃仆从表中寻找对象,找到则使用相应的仆从(即相应地伺服程序)。如果 POA 未从活跃仆从表中找到一个 Servant 与请求中的 ObjectId 值相关联,或者 POA 的保留策略为 NO_RETAIN,POA 将会采取如下行动:

①如果 POA 的请求处理策略取值为 USE_OBJECT_MAP_ONLY,那么 POA 就引发 OBJECT_NOT_EXIST 系统异常;

②如果 POA 的请求处理策略取值为 USE_DEFAULT_SERVANT,并有一个 DefaultServant(默认伺服)已与 POA 相关联,那么 POA 就调用 DefaultServant 的适当方法。如果没有 DefaultServant 与 POA 相关联,POA 就引发 OBJ_ADAPTER 系统异常;

③如果 POA 的请求处理策略取值为 USE_SERVANT_MANAGER,那么就要调用与 POA 相关联的仆从管理器,用来寻找并返回一个实现该请求的仆从。

2　POA 的策略

POA 的策略可以用来控制调用的服务质量。一个给定的 POA 支持一个或多个策略(Policy)[6]。一个策略是一个和某个应用相关的 POA 对象。不同的 POA 可以有不同的策略,从而具有不同的行为特征。CORBA 规范中的 POA 有关的 7 种策略对象是:生命周期策略 Lifespan-Policy;标识符唯一性策略 IdUniquenessPolicy;对象标识符分配策略 IdAssignmentPolicy;伺服程序保留策略 ServantRetentionPolicy;请求处理策略 RequestProcessingPolicy;隐式激活策略 ImplicitActivationPolicy;线程策略 ThreadPolicy。

下面简单介绍一下这些策略:

(1)生命周期策略。处理对象的生命期事项。生命期可以是"短暂的",也可以是"持续的"[7]。

(2)标识符唯一策略。用来确定同类对象的不同实例是否可以共用一个伺服程序。

(3)对象标识符分配策略。这个策略定义对象 ID 由谁来定义,是由 POA 产生,还是由用户来选择。需要注意的是无论是哪一种分配方法,都要保证对象 ID 的唯一性。

(4)伺服程序保留策略。CORBA 提供了 2 种方式来控制伺服程序的管理:保留方式(RETAIN)和不保留方式(NON_RETAIN)。

(5)请求处理策略。此策略的选择直接影响到 POA 调度请求的效率和系统对内存的管理。由上述可知,CORBA 提供了 3 种 POA 处理请求的方式:即使用激活的对象映射表方式、使用伺服管理器方式和使用默认伺服程序方式。

(6)隐式激活策略。激活是指把 CORBA 对象和具体化它的伺服程序关联起来,以供 POA 调度请求使用。CORBA 提供两种激活方式:隐式激活(IMPLICIT_ACTIVATION)和显示激活(NO_IMPLICIT_ACTIVATION)。

(7)线程策略(ThreadPolicy)。CORBA 给 POA 提供了 2 种线程模型:ORB 受控模型(ORB_CTRL_MODEL)和单线程模型(SINGLE_THREAD_MODEL)。

3　利用策略改善

POA 灵活性和复杂性的重要体现就是 POA 上可以进行灵活的策略组合设置。不同的应用要求需要通过不同的策略组合来表达。因此策略选择的好坏将直接影响到 POA 的性能,进而影响整个 CORBA 性能的发挥。以下提出了五种 POA 策略的组合情况的讨论,用户可以分步选择生存期策略、对象标识符分配策略、标识符唯一策略、隐式激活策略、伺服程序保留策略。那么,如何合理有效的组合这五种策略呢?

从实际应用的角度出发,对此五种策略分析比较如下:

(1)生存期策略的选择。在实际应用中如果你需要让对象的生存期超过创建它的服务器进程的生存期,这时你可以选择持久对象[8]。由于持久对象还必须对对象状态持久存储加以维护,耗费了一定的系统资源,故在没有必要的情况下,有些对象的生存周期可以设置为"短暂的"。如面向会话(session－oriented)的应用程序一般选择暂态对象,因为它们的大多数对象的生存时间仅与每个客户会话持续的时间一样长。可见,对象生存期策略的选择要根

据实际应用的需要来决定。

(2)对象标识符分配策略的选择。对于持久对象,为了便于理解和使用,应采用用户分配对象 ID 方式,由用户选择一个有意义的字符串作为对象 ID;而对于暂态对象,因为它的生存时间是短暂的,对象 ID 可以由系统自行分配一个无联想意义的字符串。当然,用户分配对象 ID 方式也可以使用。可见,通常对象 ID 的分配和解释主要还是由应用程序开发者来完成。

(3)标识符唯一策略的选择。从整体上讲,这是一种时间和空间的折中;是占用更多的内存,还是让服务器以更快的速度来处理请求。一般而言,每一个对象都会有自己的成员变量(对象状态),而在一个伺服程序中无法保存多个对象状态。因此当选择暂态对象时,建议使用唯一映射,使每个暂态对象的状态保存在对应的一个伺服程序中;当选择持久对象时,就可以使用多样映射,此时对象的状态可以保存在持久存储中。当然,为了提高处理请求的速度,持久对象也可以使用唯一映射,但如果系统同时有成千上万个对象被激活,则往往需要大量的内存来存放相应的成千上万个伺服程序,这样将会占用大量的内存空间。故对象标识符策略的选择和生存期策略的选择有着很大的关系。

(4)隐式激活策略的选择。由于隐式激活是使用映射语言提供的快捷函数实现激活的,故选择隐式激活的优点在于方便快捷,使用起来比较简单。但它也有一个缺点,容易意外地创建对象。所以当选择持久对象时,应使用显式激活,因为意外地创建持久对象将对应用程序产生不良的后果;而当选择暂态对象时,则可以使用隐式激活,因为意外产生的暂态对象终将随着服务器进程的消亡而消失,不会对应用程序产生太大的不良后果。当然,暂态对象也可以使用显示激活。

(5)伺服程序保留策略的选择。当选择保留方式时,对应的伺服管理器是伺服程序激活器。这种方法没有把所有的对象都保存在映射表中,而是通过激活器对象来动态地创建伺服程序并在 POA 中注册。如果应用程序每次仅是对同样的一些对象进行反复调用,采用这种方式在提高查找效率的同时也节约了内存空间。另外,激活器对象还提供了一个释放伺服程序的函数(etherealize),这个函数将有效地帮助用户管理内存空间,防止不必要的内存泄漏。然而在极端的情况下,所有的对象都被激活了,不仅耗费了大量的内存空间,POA 还需要大量的时间来找到目标对象的伺服程序。当选择不保留方式时,对应的伺服管理器是伺服程序定位器。由于它不保留映射表,所以不存在查找效率的问题,存储需求也达到最小,因为随着客户请求的完成系统将自动调用定位器对象的相应函数释放内存空间。然而请求的时间开销很大,而且对一个对象的多次请求需要多次创建伺服程序,浪费了一定的时间和系统资源。从应用的角度来看,此策略的两种取值各有其优缺点,需要根据实际的具体要求来设置。二次开发者可以通过选择伺服程序保留策略值来控制伺服程序在内存中的分配情况。

总之,这五种策略相互制约,互相配合。根据对以上这五个策略的分析和比较,提出四组在实际应用中较为合理的 POA 策略组合方案如下:

方案一:持久对象 + 用户分配 + 多样映射 + 显式激活

方案二:暂态对象 + 系统分配 + 唯一映射 + 隐式激活 + 保留方式

方案三:暂态对象 + 系统分配 + 唯一映射 + 显式激活

方案四:暂态对象 + 用户分配 + 唯一映射 + 显式激活

此外,还要注意具有 USE_DEFAULT_SERVANT 策略值的 POA 要求 MULTIPLE_ID 策略值的支持。具有 USE_ACTIVE_OBJECT_MAP_ONLY 策略值的 POA 要求 RETAIN 策略值的支持。而具有 NON_RETAIN 策略值的 POA 要求 USE_DEFAULT_SERVANT 或 USE_SERVANT_MANAGER 策略的支持。

4 结 论

OMG 推出的 POA 规范定义了一组组成构件及一组策略,很好地实现了分布式的计算机环境中接受客户请求及定位对象实现的功能,策略选择的好坏将直接影响到基于 CORBA 的分布式应用程序的优劣。文中从实际应用的角度出发,提出了一套新的 POA 调度策略,二次开发人员能够根据具体要求的不同设计应用程序,应用表明它是行之有效的。

参考文献:

[1] Steve V. CORBA: Integrating Diverse Applications Within Distributed Heterogeneous Environments[EB/OL]. 1999 - 02/2004 - 03. http://www.iona.com/hyplan/vinoski/ieee.pdf.

[2] 杨帮青,徐学洲. CORBA 中 POA 的分析及优化[J]. 电子科技,2002(3):11 - 14.

[3] 张 漫,陈冬芳,向明尚,等. CORBA 中多 POA 机制的探讨与研究[J]. 大庆石油学院学报,2005,29(2):68 - 69.

[4] 朱其亮,郑 斌. CORBA 原理及应用[M]. 北京:北京邮电大学出版社,2001.

[5] 刘 铁,李志蜀,许 雷. POA 剖析及实现[J]. 计算机工程,2002,28(3):279 - 281.

[6] Romer A P K. MICO - 开源 CORBA 的实现[M]. 北京:中国电力出版社,2001.

[7] 衷璐洁. POA 的 CORBA 应用研究与 JAVA 实现[J]. 计算机应用研究,2002,19(1):99 - 102.

[8] 王 麒,王晓东,傅清祥. CORBA 中 POA 策略的剖析和应用[J]. 福州大学学报,2001,29(4):24 - 27.