计算机科学2001Vol.28№.7

RECOM:一个反射中间件

RECOM:A Reflective Middleware

杨思忠　骆志刚　刘锦德

(电子科技大学计算机科学与工程学院　成都610054)

Abstract Current middlewares are limited in their flexibility and adaptability in face of changing environment and different user requirements. Applying the reflection technology to the middleware design has become a new research field. In this psper, the processing of middleware and the reflection mechanism are compared, and then the reflecive view of middleware is yielded. Based on this, the idea of employing binding reification reflective model in middleware design is proposed, and the design principles of a reflective middleware prototype named RECOM are deduced. Whereafter, this paper details RECOM's implementation about its binding factories, refletive structure, and configurable reflective layers.

Keywords Reflection, Middleware, Binding-reification, Reflective layer

1. 引言

中间件处理的是复杂的分布式应用问题,常常面对变化的运行环境和不同的用户需求。当前的基于对象的中间件,无论 CORBA,DCOM 还是 Java RMI 基本上都采用了黑箱抽象的设计原则,即对外呈现其功能而隐藏其内部实现,这使这些中间件缺少必要的灵活性和适应性[1]。

比如一个便携式计算机从局域网环境移动到无线环境,则可用带宽或传输出错率肯定会发 生明显变化。理想上希望中间件能切换到一种节约带宽的机制,如对调用消息进行压缩,但大多数中间件不支持这种适应性,而应用受限于只能使用中间件提供的机制。

分布式环境中的不同应用常常有不同的服务质量要求(如性能、安全、可靠性、负载平衡等),但中间件对此要么支持很少,要么将各种属性交织在一起。比如 CORBA 的对象服务,每个对象服务都为核心 ORB 拓展了额外的属性,但是无论这些服务使用与否,ORB 必须包含对所有服务的数据结构和协议的支持。这样使 ORB 的实现低效,用户不得不涉及事实上不需要的一些属性。

由于缺乏中间件的支持,应用程序往往不得不自己实现这些非功能性属性,这样,服务质量要么被忽略要么难于实现,要么其实现与应用程序紧密结合,而不能在其它应用中重用。

我们需要中间件的实现更灵活、更开放,采用反射技术[2],可以使系统具备检查和调整自身行为的能力。在计算机学科中,反射技术的研究和应用已从最初的编程语言,扩展到窗口系统、操作系统和分布式系统等领域。对于中间件中的反射技术的研究才刚刚起步,但已引起了各国研究者的关注。在 Middleware 2000会议上,与会学者就针对反射中间件(Reflective Middleware)进行了专题讨论[3]。

本文首先介绍了反射技术;然后将中间件的工作过程与反射机制进行比较,得出中间件的反射视图;据此,提出了采用绑定具体化反射模型来设计中间件,并进一步分析了一个反射中间件的原型 RECOM (REflective COnfigurable Middleware)的设计思路;接下来,文章从绑定生成器、反射结构、可配置的反射层等方面介绍了 RECOM 的实现。

2. 反射技术

反射,抽象地说,是系统的一种推理和作用于自身的能力。反射系统,是指这样一种系统:它提供了关于自身行为的表示,这种表示可以被客户检查和调整,且与它所描述的系统行为是因果相联的。因果相联,意味着对自表示(self-representation)的改动将立即反映在系统的实际状态和行为中(这也是使用"反射"这个词的由来),反之亦然。因此,可以简单地说,反射系统是支持因果相联的自表示(causally connected self representation,CCSR)的系统[2]。

就如在传统的面向对象编程中,对象代表了现实

杨思忠　博士生。骆志刚　博士生。刘锦德　教授,博导。

世界中的事物一样，对象本身（其状态、行为等）也可用其它对象来表现，后者又称为元对象。元对象的计算（又称为元计算）用于观测和修改它们的指示物（即所代表的对象）。元计算常常是通过捕获它们的指示物的正常计算、由元对象执行的；换句话说，指示物的行为被元对象所捕获，后者执行元计算，要么替换要么封装指示物的行为。当然，元对象自身也可由其它对象来表现，即它们是二重元对象（meta-meta-object）的指示物，以此类推。这样，一个反射系统可以是一种多层结构，组成了一个反射塔。在基层的对象，对应用领域中的实体进行计算；其它层，即元层的对象，对其相邻低层（指示物层）的对象执行计算。

每一次反射计算可以被分为两个逻辑部分：计算流上下文切换和元行为。计算从基层的计算流开始；当基层实体执行某个行为时，该行为被元实体捕获，同时计算流上升到元层（称之为换上操作）；然后元实体执行完它的元计算，当它允许基层实体执行时，计算流又返回到基层（称之为换下操作）。

在所有反射模型中，一个基本概念就是具体化（reification）。为了对相邻低层的计算执行计算，每一层维护一组支持这个计算的数据结构，即前述的因果相联的自表示，而创建这种自表示的过程就称为具体化。具体化就是将原本对客户隐藏的方面显式化。当然，相邻低层的哪些方面被具体化，取决于反射模型（如：结构、状态和行为、通信等）。在文[4]中，根据元实体与基层实体的关系，首次对反射模型进行了分类，指出了三类反射模型：元类模型、元对象模型和元通信模型。在任一种情况下，包含自表示的数据结构与系统被具体化的那些方面是因果相联的。在相邻层之间保持这种因果相联关系是反射基础设施的责任，而元对象的设计者和编程人员不必知道如何实现因果相联关系等细节。

反射模型的一个关键特色就是透明性[6]。在反射这个上下文中，透明是指位于任一层的对象完全不知晓其上层对象的存在或工作情况。换句话说，每个元层被加到其指示物层上而无需改动指示物层本身。也即，反射系统在元层及其指示物层之间执行因果相联，是以对元层编程人员和指示物层编程人员都透明的方式实现的，我们把这称为反射透明性。

3. 中间件的反射视图

基于中间件的分布式应用中，在不同地址空间的两个对象需要进行交互时，客户首先获得一个服务器的代理（所谓代理就是代表远程对象的一个本地对象），代理具有与所代表的远程对象相同的界面。客户通过代理调用服务器的方法，代理再利用中间件的通信机制将调用请求传递给目标服务器，得到结果后，代理再将结果返回给客户，这样就完成了一次远程方法调用。而在调用之前，将系统中的两个对象联系起来（以实现交互）的过程，及此过程形成的结果，称为绑定。

仔细审查上述中间件的工作过程，发现与前述的反射过程非常相似。可以作以下类比：基层实体执行某个行为（客户与服务器之间进行交互），该行为被元实体所捕获（客户并不是直接调用服务器，而是由中间件在网络中传递调用消息，代理起着捕获基层行为的作用），计算流上升到元层；然后元实体完成它的元计算（对调用消息进行编码，通过网络传送到服务器端；对参数等解码后，调用服务器；得到结果后，对结果进行编码，然后传回客户端，再对结果解码）。当它允许基层实体执行时（代理将调用结果返回客户），计算流又返回到基层。

这样可以很自然地将中间件建立的客户与服务对象之间的绑定，设计为由若干不同功能的元对象组成。即将绑定具体化，作为中间件系统的自表示，可由应用程序检查和调整，以满足应用的不同服务质量要求。图1表示了中间件的反射视图。

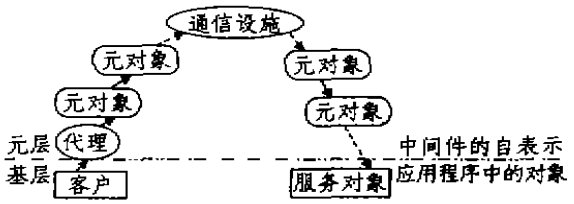

图1　中间件的反射视图

这个反射模型与前面提到的三种反射模型都不一样，它具体化的是分布对象之间的绑定，可称为绑定具体化反射模型。而那三种模型具体化的是一个对象（不是对象之间）的行为。我们采用绑定具体化反射模型，实现了一个反射中间件的原型 RECOM。现在可以有意识地运用反射技术的思想，来考虑 RECOM 的设计。

4. RECOM 的设计思路

·自表示的改动将立即反映在系统的实际状态和行为中，这里的自表示就是分布对象之间的绑定，可以考虑绑定是有类型的，一个绑定的类型代表了所用的调用语义（同步或异步）、协议、可能的服务质量限制或其它的绑定属性。这样用户可以从多种不同类型的绑定中进行选择，有时还可能对被选中的绑定进行改动（配置）。

·反射基础设施要支持反射透明性。这要求元层

(中间件的自表示)对基层实体(应用中的对象)是透明的,这与一般中间件中的访问透明性是一致的,另一方面,要求进行元计算的各元实体与相应的基层实体的具体类型是无关的,即希望构成绑定的模块是通用的,可以对不同类型的基层实体进行元计算。

·具体化是将系统原本隐藏的方面显式化,即自表示可以被用户检查和调整。根据应用的不同服务质量的要求,用户可能要改动绑定,或另外设计和配置其中的模块,这也要求各模块支持通用的界面;另外需要对用户开放的配置机制,因此要设计一个显式绑定协议。

5. RECOM 的实现

RECOM 是基于 Java 的,Java 既是一种面向对象的语言,又是一个平台。在 Java 平台上构造中间件,可以利用 Java 虚拟机屏蔽下层异质的操作系统和物理机器。特别是 Java 的结构反射[6]、动态滞后连接、对象串行化、网络编程等特性,从多方面简化了 RECOM 的设计。

5.1 绑定生成器

为了能被远程访问,服务器首先输出其界面引用。客户获得这个界面引用,根据其中信息,与服务器建立绑定,然后进行交互。在 RECOM 中,绑定是由一类特殊的实体——绑定生成器构造的。绑定生成器主要有两方面的作用:一是生成和管理界面的引用;二是创建和管理特定类型的绑定。

不同的绑定生成器构造不同类型的绑定。一个绑定的类型代表了所用的调用语义(同步或异步)、协议、可能的服务质量限制或其它的绑定属性,每个远程界面引用包含如下信息:引用所代表的界面的类型;生成引用的绑定生成器的种类;还必须包含一些标识信息,用于在网络中唯一地确定被引用的界面,标识信息的表达与具体的绑定生成器所用协议有关,例如,使用基于 TCP/IP 的远程调用协议(如 IIOP 协议族),标识中通常包括一个主机名字和一个端口号。

RECOM 的结构,可以看作是由 ORB 核和一组绑定生成器组成。服务器选用适当的绑定生成器,生成其界面引用,然后将该引用输出(如输出到某个目录服务);客户获得这个界面引用后,利用同类型的绑定生成器对其进行解析,在客户与服务器之间建立绑定,并获得该服务器的本地代理(桩)的引用。之后,客户就可以通过代理与服务器进行交互。下面是一个绑定生成器的基本界面:

```
interface BindingFactory
{
    Reference generateReference(Object obj,Class cls);
    void dropReferences(Class cls);
    Stub resolveReference(Reference ref,Class cls);
    String stringifyReference ( Reference ref ) throws-
        BadReference;
    Reference parseReference(String sref)throws BadRef-
        erence;
}
```

图2 绑定生成器的基本界面

其中,dropReferences 操作放弃界面 cls 的所有引用,这样客户不能再通过这些引用访问实现界面 cls 的对象。stringifyReference 和 parseReference 用于界面引用与其字符串形式之间的转换。

5.2 反射结构

在 RECOM 中,远程对象的本地代理,是一个简单的客户桩对象,用于将特定类型的调用转化为通用形式(类 Invocation 的一个实例),并将调用请求传递给协议栈的最上层。相对其它系统而言,RECOM 的客户桩比较简单,因为它不负责调用请求的本地语言表示与用于网络传输的字节流之间的转换,即所谓的编码/解码功能,在 Java 中又称为串行化/逆串行化。而且,RECOM 的客户桩是与下层协议无关的,也就是说同样的客户桩可以和不同的协议族结合使用。由于简单,客户桩类是在客户进程中自动生成(这是利用 Java 的结构反射功能对调用界面进行分析而完成的),然后动态连接到程序中。

RECOM 的协议栈与其它系统相比要复杂,它负责调用的各个方面。除了如对调用参数进行编码,执行远程方法调用协议等操作外,还可以包括一些高层特性,如复制的管理、调用结果的缓存等。

RECOM 的通信协议栈设计为一组元对象,每个元对象,将调用作为一个数据结构依次进行操作,直到最后调用远程目标对象。这里也利用了 Java 的结构反射,在服务方不再需要桩(stud)或构架(skeleton),而是在调用控制层,利用调用中对被调用方法的描述,直接调用目标对象。

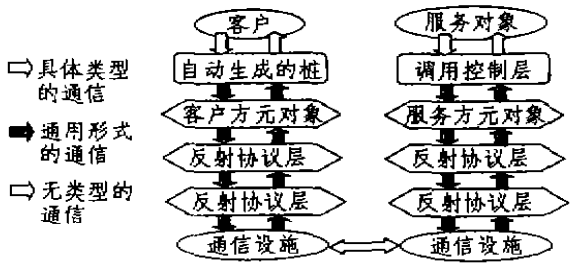

图3 RECOM 的反射结构

图3表示了一系列元对象组成通信协议栈。元对象是通常的 Java 对象,因而也完全是类型安全的。协议栈往往分层实现,所以又将其中的元对象称为反射层,这种形式的反射是协议相关的。由于不同协议栈有许

多共性的东西，在设计一个反射层时可考虑通用性，使其可在多个协议栈中使用，以提高代码的重用性。

在客户方协议栈的顶部，一个调用由目标界面的引用、被调用的方法和表示为一组对象的调用参数组成。界面引用是一个复杂的对象，协议栈根据界面引用中的信息，确定从客户到目标界面的访问路径。图4描述了类 Invocation 的基本界面。

```
class Invocation
{
    Invocation(WrappedMethod method,Object target,Object
        arg[]);
    Object          getTarget();
    void            setTarget(Object target);
    Method          getMethod();
    Void            setMethod(Method m);
    Object[]        getArguments();
    void            setArguments(Object[] o);
    Object          getReturnValue();
    vtod            setReturnValue(Object o);
    void            setExceptionalReturn(Throwable t);
    boolean         isExceptionalReturn();
}
```

图4　类 Invocation 的基本界面

当调用在栈中行进时，每一层都可以对调用进行处理。到达栈的底层时，最初的目标界面引用已被解析为适当的网络端点或连接标识符，而且缓冲区中已串行化了足够的信息以在服务器重新构造该调用，在服务器上，进行相反的处理，最终得到被调用的目标对象、方法和参数等信息。

5.3　可配置的反射层

RECOM 的协议栈相对传统的中间件系统而言要复杂，因为它负责调用处理的各方面。除了基本功能，如：对参数进行编码、执行 RPC 协议等外，还可包括高层特性，如：管理复制、调用结果缓冲、加密、压缩等等。

但是，由于 RECOM 利用了反射技术，首先由客户桩把具体类型的调用转化为通用形式，然后将各协议层看作是对调用进行处理的反射层，并实现统一的界面，因此协议栈的设计变得简单、灵活，而且提高了代码的可重用性。可以设计配置多种类型和性能的协议栈，如在传输层可以用 UDP、TCP，也可用实时的 RTP；消息层可针对不可靠的网络传输使用 REX 协议[7]，也可使用简单的请求响应协议；在表示层可选用 XDR 或 CDR 等多种编码方案。如果用 CDR 进行编码、在消息层使用 GIOP、使用 TCP 进行网络传输（采用 IIOP 协议族[8]），则可以实现 RECOM 和 CORBA 中间件的互操作。这些不同类型或不同服务质量的协议栈由相应的绑定生成器创建和管理。

不仅如此，RECOM 还允许在运行期，对特定客户与服务器间的绑定进行配置，在协议栈中插入一些特殊功能的反射层。这些特殊的反射层实现中间件的一些非功能性属性，而且是通用的，可配置在不同的协议栈中。它们可以分为两类：请求级反射层和消息级反射层。请求级反射层，插入在客户方的桩与编码层之间，或服务器方的调用控制层与编码层之间。它截取的是类型化的调用信息，可用于对调用结果进行缓存，以提高性能；在访问当前服务器失败的时候，自动将调用重定向到提供相同功能的其它服务器，实现高可用性等。消息级反射层，插入在客户方或服务器方的 PRC 层与传输层之间。它截取的是已被编码成字节流形式的调用请求，可用于对调用消息进行加密/解密、压缩/解压等，RECOM 提供了一些可选用的反射层，用户也可以根据两类反射层的界面，实现自己的特定功能的反射层。

为了对绑定进行配置，RECOM 提供了一个显式绑定协议。通过这个显式绑定协议，客户（或第三方）不仅可以获得调用服务器的引用，还可以得到客户方绑定的引用和服务方绑定的引用。也就是说，绑定成为第一类的对象，可以被用户操作（配置）。按照反射技术的术语，即绑定被具体化了。图5给出了绑定的配置界面。

```
Interface BindMgr
{
    ID addReqRftLayer(ReqRftLayer rlayer);
    Boolean removeReqRftLayer(ID id);
    Boolean removeAllReqRftLayer();
    ID addMsgRftLayer(MsgRftLayer mlayer);
    Boolean removeMsgRftLayer(ID id);
    Boolean removeAllMsgRftLayer();
}
```

图5　绑定的配置界面

值得指出的是，对于具体某个绑定，可以同时配置多个反射层。

总结　CORBA2.2规范引入了截取器的概念[8]。截取器允许用户监控调用请求和响应，既可用于客户方，也可用于服务器方，规范定义了两种不同类型的截取器：请求层截取器和消息层截取器。但是，CORBA 的截取器是通过回调的形式实现的，与 RECOM 的反射层相比，最大的局限性在于它不能改变中间件系统中消息处理的基本控制流。而反射层得到调用信息，进行处理后，要显式地调用调用链中的下一层。所以，它可以捕获中间件、其它反射层或服务对象的异常；它可以对这些异常自己进行处理，也可重新抛出；由于是显式调用下一层，因此也可以完全不调用下一层，而是对调用请求作本地处理后，就直接返回（如在客户方缓存了以前的调用结果）。

可以看出，通过提供不同类型的绑定生成器；并在运行期，允许用户对具体客户和服务器之间的绑定进行配置，RECOM 具有较好的灵活性和适应性。而且，利用反射层可以实现一些非功能性属性，如高可用性、安全、负载平衡等。而这些反射层是可配置的，因此

RECOM 还将中间件的功能性属性(远程方法调用)与非功能性属性相分离。这些都得益于反射技术的运用。

反射技术在中间件中的应用研究才刚刚起步，Illinois 大学的 dynamicTAO 项目[9]，Lancaster 大学的 OpenORB 等项目[10]，在这方面进行了开创性的研究。但是 dynamicTAO 和 OpenORB 都是将 ORB 的某些内部实现开放出来。我们将绑定作为中间件的自表示，而每个绑定完全监控了分布式对象间通信的全过程，换句话说，也就是这种自表示完全地反映了中间件实现的各个方面。我们采用绑定具体化反射模型来设计中间件，从而推导出具有指导意义的设计思路，并实现了一个反射中间件的原型 RECOM。

进一步的工作包括：丰富 RECOM 的绑定生成器和反射层；将 RECOM 运用到一个实际应用中；将实现和配置反射层的代码(按反射技术的术语即元程序)从应用程序分离出来，仅在需要的时候才与应用程序发生关联，因为编写元程序需要专门的知识，可由专人完成；研究不同反射层之间的制约关系；等等。

参 考 文 献

1 Kiczales G. Towards a New Model of Abstraction in Software Engineering. In. Proc. of the IMSA'92 Workshop on Reflection and Meta-level Architecture, 1992

2 Maes P Concepts and Experiments in Computational Reflection. OOPSLA'87, Sigplan Notices, Oct. 1987

3 RM'2000 — Workshop on Reflective Middleware. 2000. Available at: http: // www. comp lancs. ac uk/computing/RM2000

4 Ferber J. Computational Reflection in Class-based Object Oriented Languages. OOPSLA'89, Sigplan Notices, Oct. 1989

5 Cazzola W. Evaluation of Object-Oriented Reflective Model. In. Proc. of ECOOP Workshop on Reflective Object-Oriented Programming and Systems (EWROOPS'98). Brussels, Belgium, Jul. 1998

6 Sun Microsystems. Java Core Reflection Available at: http. // Java. sun. com/products/jdk/1. 2/docs/guide/reflection/index. html

7 Hayton R. ANSA Team FlexiNet Architecture. Architecture Report, Citrix Systems (Cambridge) Limited, Feb. 1999

8 The CORBA Architecture and Specification, Reversion2. 2 Object Management Group, Inc. Feb. 1998

9 Roman M, Kon F, Compell R. Design and Implementation of Runtime Reflection in Communication Middleware: the Dynamic TAO Case. In: Proc. of the ICDCS'99 Workshop on Middleware Austin, Texas, May 1999

10 Costa F, Blair G. A Reflective Architecture for Middleware: Design and Implementation. In. Proc. of the ECOOP'99 Workshop for PHD Students in Object Oriented Systems, Lisbon, Jun. 1999

(上接第97页)

不完全演化重复次数在10～50之间，搜索空间收缩迭代过程中，搜索空间边长按等比缩小，不完全演化重复次数按等差减小。$f_1(x)$的50个结果中两个结果在6500～6000之间，其余的均小于6000，表1是各种算法对 $f_1(x)$的计算结果比较。$f_2(x)$的50个结果都小于1.9，表2是各种算法对 $f_2(x)$的计算结果比较。

结论 本文为解约束优化问题提出了一种新的演化算法。在这个方法里，演化算法不完全地重复执行提供问题较优解的分布信息，利用该分布信息，本算法定位最优解的可能范围，逐渐缩小搜索空间。在这种算法中，演化算法既用来定位最优解区域，实现搜索空间自动向最优解收缩，又用来最终求解最优解。2个测试问题的计算结果表明本文的算法在解的质量、稳定性和收敛速度等方面优于一般演化算法。

参 考 文 献

1 Michalewicz Z, et al. Evolutionary algorithms for constrained engineering problems. Computers & Industrial Engineering J., 1996, 30, 851～870

2 Cao Y J, Wu Q H. Mechanical Design Optimization by Mixed-Variabel Evolutionary Programming. In: Zbigniew Michalewicz, Xin Yao, eds. Proc. of the 1997 Int Conf. on Evolutionary Computation. Indianapolis Indiana: IEEE Sevice Center, 1997. 443～446

3 Dannan B D, Kramer S N. An Augmented Lagrange Multiplier Based Method for Mixed Integer Discrete Continuous Optimization and Its Applications to Mechanical Design. *Journal of Mechanical Design*, *1994*, *116*: *318～320*

4 Kalyanmoy Deb. DeneAS: A Robust Optimal Design Technique for Mechanical Component Design. In: Zbigniew Michalewicz, eds Evolutionary Algorithms in Engineering Application. Berlin: Springer-Verlag, 1997. 497～514

5 Coello C A. Self-Adaptive Penaltres for GA-based Optimization. In: Angelme P, ed. Proc. of the 1999 Congress on Evolutionary Computation Washington D. C. USA: IEEE Service Center, 1999, 3: 2159～2169

6 Ragsdell K M, Phillips D T. Optimal Designed Class of Welded Structures Using Geometric Programming. ASME Journal of Engineering for Industrial, 1976, 98(3), Series B. 1021～1025

7 Lalyanmoy Deb. Optimal Design of a Welded Beam by Genetic Algorithms AIAA Journal, 1991, 29(11): 2013～2018