一种基于 CORBA 的分布式应用模型①

李海闻 宁 敏 林福良 周 武 (北京控制与电子技术研究所

C4ISR 技术国防科技重点实验室 北京 100038)

摘 要: 在对 CORBA 通信机理研究的基础上,提出了一种以 CORBA 技术为基础面向对象的分布式应用模型。该模型通过封装 CORBA 的复杂性,降低开发分布式应用的难度,提高开发效率,提供分布式对象之间的透明访问。以中间件的形式实现了这个模型并做了测试。

关键字: CORBA; 中间件; 分布式对象计算

A Distributed Application Model Based on CORBA

LI Hai-Wen, NING Min, LIN Fu-Liang, ZHOU Wu

(C4ISR Technology National Defense Science and Technology Key Lab, Beijing Institute of Control and Electronic Technology, Beijing 100038, China)

Abstract: On the basis of researching CORBA communication mechanism, this paper brings forth an object-oriented distributed application model based on CORBA technology. This model reduces difficulty and improves efficiency on distributed application development, through encapsulating the complexity of CORBA. The model provides transparent requests among distributed objects. The model is realized with middleware form and tested.

Keywords: CORBA; middleware; distributed object computing

在分布式计算中,中间件是一类介于操作系统与应用之间的一层起承上启下作用的支撑软件,通过建立分布式软件模块之间的互操作机制,屏蔽底层环境的复杂性和异构性,为上层的应用软件提供运行和开发环境,帮助用户灵活、高效地开发和集成复杂的应用软件。面向对象的中间件技术是中间件平台的主流技术,OMG 的 CORBA 分布计算技术是其中的主要代表之一。

CORBA 是 OMG 组织制定的公共对象请求代理体系规范,是一种异构平台下语言无关的网络分布式对象互操作模型,是开发分布式应用的中间件技术。但是 CORBA 涉及到的技术复杂、庞大,使学习和掌握它的时间周期较长,不利于应用的快速开发。在对 CORBA 原理及异步消息通信机制[1,2]研究的基础上,本文提出了一种以 CORBA 技术为基础面向对象的分布式应用模型,该模型通过封装 CORBA 的复杂性,降低开发分布式应用的难度,提供网络中对象之间的透明访问,便于快速开发应用。

1 CORBA中的几个重要概念

本文的模型设计以 CORBA 技术为基础,其中用到了一些 CORBA 的概念和术语,为了使后面的描述更清楚这里先简要介绍 CORBA 的几个概念。

1.1 对象请求代理 ORB(ObjectRequestBroker)

CORBA 是一种异构平台下语言无关的分布式对象互操作模型,其核心就是 ORB。ORB 是对象请求与接受响应的机制,它作为分布式对象之间的通讯中介,为定位对象、激活对象提供方法。ORB 的基本功能就是支持访问异地对象。

① 基金项目:国防基础科研项目(K0402010205)
收稿时间:2009-05-03

1.2 对象引用

对象引用是一个用来标识、定位 CORBA 对象的句柄。对象引用是客户与服务器程序之间的纽带，客户要访问服务器程序中的对象，首先必须拥有该对象的对象引用，对象引用中包含了定位服务器程序及服务对象的所有信息，标识唯一的目标对象。

1.3 名字服务

名字服务是一个基本的 CORBA 对象服务，它把名字映射为对象引用。这种“名字-引用”二元组合称为名字绑定。通过把对象引用保存在名字服务中，应用程序可以通过逻辑名来得到 CORBA 对象引用。名字上下文是保存名字绑定的对象，也就是一个从名字到对象引用的映射表。客户端可以通过名字上下文枚举映射表中名字绑定。

2 模型描述

该模型结构上分为三层，底层以 CORBA 分布式对象请求技术为基础，通过 CORBA 对象及对象引用的访问机制为网络中各节点的数据交互建立通讯基础；中间层管理底层的 CORBA 对象及对象引用和上层的应用对象，为上下两层提供双向的数据定位功能，并为上层封装注册、访问等接口；上层是应用对象层，应用对象以服务名的形式向中间层注册，为网络中的其它应用对象提供服务。

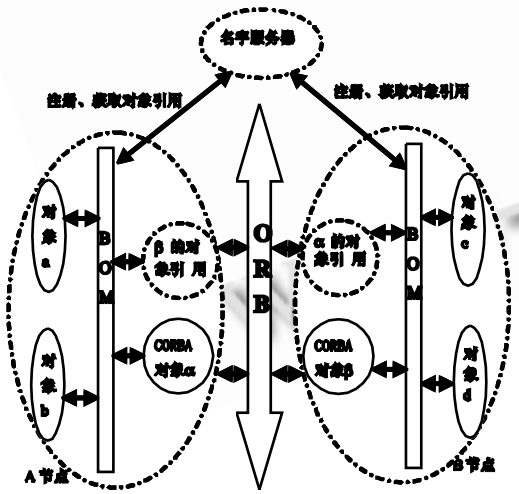

图 1　模型结构示意图

具体地，每个节点上都驻留一个 CORBA 对象，作为本节点对外提供服务的代理对象，如图 1 中的对象 α，对象 β。节点之间底层的数据交互是通过这个代理对象及其对象引用来实现的，也就是 A 节点通过 β 的对象引用访问 B 节点中的 β 对象，B 节点通过 α 的对象引用访问 A 节点中的 α 对象。各节点上的代理对象负责接收服务请求并通过 BOM(代理对象管理器)定位到本地的某个应用对象。每个节点保存含 n-1 个元素的已注册节点的对象引用表(n 表示网络中的注册节点数量)，表中每个对象引用代表了一个节点，通过这个对象引用可以向对应的节点发送服务请求，这里的服务请求(同步、异步)可以是单向的异步数据发送，也可以是要求有返回值的同步请求。返回值在调用对象的方法时直接返回，而无需通过请求方的对象引用。一次完整的服务请求过程：A 节点应用对象 a 发出服务请求(假设被请求的对象在 B 节点)，BOM 根据被请求的服务名检索到其对应的?对象引用，请求通过这个对象引用，经 ORB 定位到 B 节点的对象 β，β 把请求交给 BOM，BOM 根据服务名检索到对应的应用对象 c，c 收到请求进行处理，结果经 ORB 由对象引用直接返回给请求方。在实现上，消息在节点内的传递是通过传递消息结构的指针来完成的，因此不会影响消息传输效率。

该模型的主要思想是：通过维护动态的网络“对象引用”表、本地“服务名—应用对象”映射表和网络“服务名—对象引用”映射表，方便灵活地实现分布式对象的透明访问。

3 模型实现

3.1 CORBA 对象的设计

CORBA 代理对象是该模型底层部分的核心，代理对象负责接收所有的消息，并交给代理对象管理器(BOM)作进一步处理。用 CORBA 技术开发分布式应用，首先要用 IDL 接口描述语言定义分布式对象的接口，再通过 IDL 编译器把对象接口描述转化为 C++或其它编程语言描述的对象。代理对象的 IDL 描述如下：

```
interface CppBrokerObject {
typedef sequence <octet> RequestParam;
RequestParam servantSync(in string serviceName,
in RequestParam param);
oneway void servantAsync(in string serviceName, in RequestParam param);
}
```

其中，RequestParam 是发送请求或返回值的参数类型；servantSync 和 servantAsync 是对象的两个方法。获得该对象的对象引用后就可以通过这两个方法实现对象的同步或异步访问，同步访问有返回值。

经过 IDL 编译器编译后会生成几个类，其中一个是该对象的实现体类，在实现体类中处理收到的请求。

3.2 代理对象管理器的设计

代理对象管理器 BOM(BrokerObjectManger)是底层 CORBA 与上层应用对象之间的桥梁，它的功能是向名字服务器注册本地 CORBA 对象的对象引用，同时从名字服务器获取网络上已注册的 CORBA 对象引用。BOM 以服务名为索引维护一个本地“服务名—应用对象”映射表和一个用来存储网络中每个服务名对应 CORBA 对象引用的网络“服务名—对象引用”映射表。BOM 通过获得的各个对象引用向除自身以外的节点发送本节点所能提供的服务列表，并获取相应节点能提供的服务列表。代理对象负责接收其它节点发来的服务对象列表，并交给 BOM 管理，同时由代理对象向其它节点返回本地的服务对象列表。BOM 类的主要成员描述如下：

```
class BrokerObjectManger
{
std::list<...>* netNodesList;
std::map<…>*      localCppObjMap;
std::map <…>*     corbaObjMap;
bool  registerLocalCppObjs(…);
bool  registerNetCppObjs();
bool      registerNetNodes();unsigned      char*
 requestSync(…);
bool  requestAsync(…);
char*  receivedRequest(…);
}
```

(1) localCppObjMap：存储本地“服务名--应用对象”的映射表。

(2) corbaObjMap：存储网络中“服务名--对象引用”的映射表。

(3) netNodesList：存储网络中已注册节点的对象引用表。

(4) registerNetNodes()：从名字服务器获取网络中已注册的对象引用。

(5) registerLocalCppObjs(…)：注册本地服务对象，上层应用对象调用该函数把服务名注册到 localCppObjMap。

(6) registerNetCppObjs(…)：由 netNodesList 中的对象引用获取网络中对象的服务名，并注册到 corbaObjMap。

(7) receivedRequest(…)：代理对象收到的所有数据交由该函数根据消息类型做进一步处理。

requestSync()， requestAsync()分别是封装后对外提供的同步和异步请求接口。

3.3 应用对象基类的设计

需要对外提供服务的应用对象从基类 CppObjModule 派生即可获得发送请求、提供服务的功能。不需要对外提供服务的应用对象通过获得 BrokerObjectManger 对象也可以请求网络中的服务对象。基类 CppObjModule 的主要成员描述如下：

```
class CppObjModule
{
CppObjModule(char*name,BrokerObjectManger*
bom);
unsigned char* requestSync(char*
serviceName,unsigned char*param,int & len);
bool requestAsync(char*serviceName,
unsigned char* param,int &len);
virtual char* OnReceiveRequest(
char*source,unsigned char*param,int &len);
char* objectName;
BrokerObjectManger*  bom_t;
}
```

(1) CppObjModule(…)：构造函数，调用 BrokerObject-Manger 的接口 registerLocalCpp-Objs()把该应用对象注册到本地服务列表。

(2) requestSync(…)：发送同步请求的接口，返回值是无效结果 NULL 或有效数据 unsigned char*，长度由第三个参数 len 传出。其内部调用的是 Broker-ObjectManger:: requestSync()。

(3) requestAsync(…)：发送异步请求的接口，返回 true、false 分别表示请求发送成功或失败。其内部调用的是 BrokerObjectManger: requestA-sync()。

(4) OnReceiveRequest(…)：服务函数，应用对象把服务的内容写在派生类的重载函数里，当对象收到请求时重载函数会自动触发。若远程是同步请求，该函数的返回值就是远程 requestSync 的返回值，也就是a对象调用自身requestSync方法请求c对象时，c对象的OnReceiveRequest的函数返回值就是a对象调用 requestSync 函数的返回值。

3.4 几种消息类型

代理对象收到的所有数据都提交给 BOM，BOM 根据数据表示的不同消息类型作进一步的分析处理。模型中定义了以下几种消息类型。

(1) CBO_CLOSE ：节点关闭消息，当某节点程序退出时，会向本地保存的对象引用列表中每个元素对应的节点发送该消息，其它节点收到该消息后会从表中注销相应的对象引用及其关联的服务列表。

(2) CBO_GET：索取服务对象列表，程序启动时从名字服务器获得对象引用列表，向列表中每一个对象引用对应的节点发送该消息，对方收到后返回本地的服务对象列表。

(3) CBO_SEND ：发送服务对象列表，系统初始化时把本地的服务列表发给各个节点。

(4) CBO_DATA：上层应用对象要传输的数据，通过 BOM 解析传递给提供服务的应用对象。

(5) CBO_MON：心跳监控消息，名字服务器周期性地向各节点发送该消息，若连续收不到某节点的返回值，则注销其对应的对象引用，并通知其它节点也注销该对象引用及其对应的服务列表。

每种消息类型分别对应一个数据结构，限于篇幅这里不再叙述。

3.5 编程规则

该模型以动态链接库的形式对外提供简单的编程模式和接口，使上层应用的开发者不必了解 CORBA 的技术细节就可以快速开发分布式应用。模型内部自动生成并维护三张动态的表，但开发上层应用时无需关注这些细节，只需约定各节点上应用对象注册的服务名不同即可。各节点应用程序在初始化阶段只需要生成一个 BrokerObjectManger 对象，通过这个对象就可以对外发送请求。需要对外提供服务的应用对象从基类 CppObjModule 派生即可完成服务注册和获得发送请求的功能。

4 模型测试

测试的主要目的是为了说明该模型提高开发效率的同时，是否对 CORBA 的性能带来了显著的影响。

测试环境：硬件环境由三台惠普 HP dc7700(CPU Intel 双核 1.8G，内存 2G，千兆以太网络)组成；软件环境由 WindowsXP 操作系统、CORBA(中创软件 InforBus5.0)及 Visual C++6 开发环境组成。

测试说明：用 InforBus 和本模型分别构建一个包含多个服务对象的例子，部署在各节点上，选择其中两个节点，一个用来发送请求，一个用来服务请求，计算从发送请求到收到服务结果经过的数据传输平均延迟 T。测试方法：按不同数据包大小分别测试 1000 次，记下每次的数据传输延迟 T[i]，平均延迟 T 等于 (T[0]+…+T[999])/1000。通过测试说明该模型没有对 CORBA 的网络传输性能产生显著影响，在单次传输不大的数据量时，两者的数据传输延迟都是毫秒级的。测试中计量较精确的时间差是通过获取机器中高分辨率计数器的计数频率和两次计数器差值的方法来实现的。测试结果如表 1 所示。

表 1　测试结果

单次发送数据量(字节)	128	512	1024	2048
InforBus平均延迟	0.76ms	0.80ms	0.85ms	1.03ms
本模型平均延迟	0.86ms	0.93ms	1.03ms	1.22ms

5 结语

本文以 CORBA 技术为基础，设计了一种分布式应用模型，通过简单的实用规则和接口可以有效地提高开发分布式应用的效率。该模型具有跨平台的互操作性、可扩展性，适合构造灵活的分布式互操作系统，解决分布式异构环境下软硬件系统的互连互操作问题。同时也为分布式系统的综合集成提供了一种有效的技术途径。该模型除了可以简化分布式应用开发，还有以下特点：

(1) 平台无关性：继承了 CORBA 适应异构平台环境的特性；

(2) 服务对象位置无关性：客户应用无需知道它所请求的服务对象在网络中的具体位置，这样有利于动态部署服务对象。在系统容错设计时也便于迁移服务对象；

(3) 支持同步请求：该模型的实现可以作为一种消息中间件来使用，一般消息中间件只支持异步请求，如果需要返回值，对方必须显式地回发。该模型不仅支持这种异步请求，也支持同步请求。

参考文献

1 张志伟,隋品波,郭长国,吴泉源.分布对象中间件异步消息的研究与实现.计算机学报, 2004,27(12):1626－1631.

2 张志伟,郭长国,蔡俊亚,吴泉源.CORBA 异步消息的研究与实现.电子学报, 2004,11:1820－1823.