基于CORBA与Web Service的系统集成实例

中山大学附属第一医院信息网络科　刘建勋

摘　要：本文介绍CORBA和Web Service这两种技术在软件系统集成中的应用，并以JACORB和Axis为例演示了两者的互操作方法。

关键字：CORBA，Web Service，JacORB

Integration Practice with CORBA and Web Service

Jianxun Liu, Department of Information &Network ,

The 1st Affiliated Hospital of Sun Yat-sen University

Abstract: The basic concept of the integration using the technology of CORBA and Web Service is introduced. and a demonstration is given to show the conception with JacORB and Axis tools.

Keywords: CORBA，Web Service，JacORB

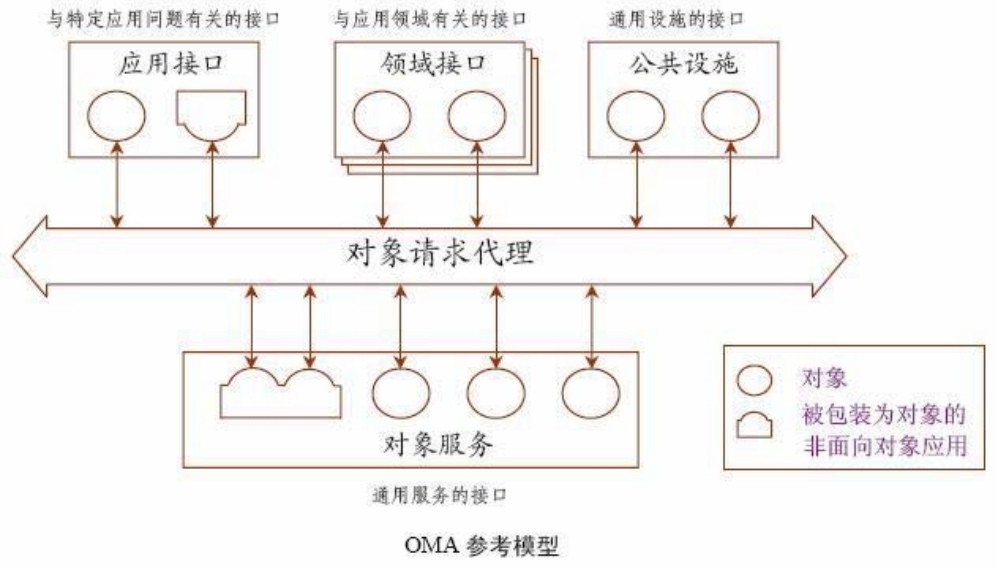
随着国家医疗卫生系统和各地方医疗保险体系的发展，各个医疗机构基本上建立了自己独立的信息系统，医疗机构间信息交互的需要慢慢凸显出来，如何在各个完全独立的信息系统之间进行数据交换，成为一个很大的挑战。由于软件开发方法的不断更新，所提供的数据交互方式也发生了很大变化，由最初的简单数据文件交换到分布式对象调用直至目前流行的面向服务的体系结构构建的分布式计算环境，各医疗机构使用的信息系统也存在较大的差异，由于这些医疗机构根据业务需求对外提供服务，同时也使用其他医疗机构提供的服务，使得数据交互的实现难度加大，为解决这个问题，这些系统各自使用了不同的实现方法，其中以OMG组织的CORBA体系结构实现的较有代表性。

1．背景知识

1．1 公共对象请求代理体系结构

CORBA(Common Object Request Broker Architecture)是对象管理组OMG(Object Management Group)这个非盈利性联盟发布的中间件规范，目的是在分布式环境下实现应用的集成，使基于对

象的软件成员，在分布和异构的环境下可重用、可移植、可互操作，其参考模型如下：



OMA 参考模型

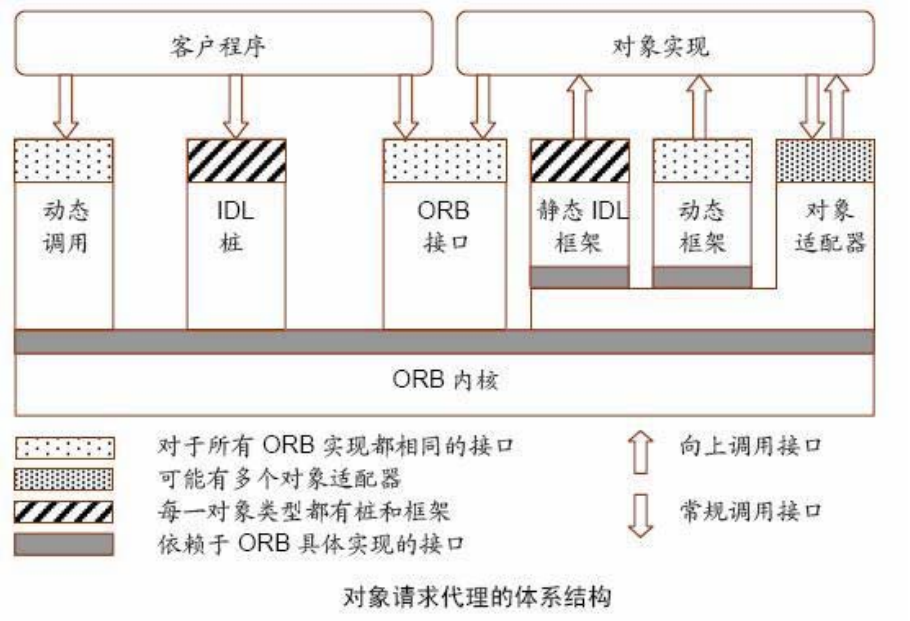
CORBA 主要由三部分组成：

① 作为分布式对象通信基础设施的对象请求代理 ORB 的体系结构；

② 接口定义语言 IDL 的语法、语义及到各种程序设计语言的映射；

③ 保证可互操作性的标准 ORB 间的通信协议 GIOP/IIOP。

其中对象请求代理 ORB 作为一个“软件总线”来连接网络上的不同对象，提供对象的定位和方法调用，它是 CORBA 实现的关键，其体系结构如下所示：



对象请求代理的体系结构

根据接口/实现分离的软件工程思想，接口的定义是服务双方共同订立的合约，服务端提供对象实现，客户端使用这些对象。对象接口使用 OMG 的接口定义语言 IDL 进行描述，IDL 根据对象可执行的操作定义对象的类型，并映射到特定的编程语言或对象系统。对象实现可以用各种不

同的编程语言实现，如 Java、C/C++、Python、Cobol、Lisp 等，同时对象可采用如服务程序、被包装的非面向对象应用程序、程序库等多种方式实现；客户程序也可以使用不同的编程语言实现，但只能通过接口定义来了解对象的逻辑结构，通过发送请求来影响对象的行为和状态。

由于 CORBA 只是一个规范，不同公司的 ORB 有各种不同的实现，不同 ORB 产品以及跟 CORBA 不兼容的分布式应用系统（如 EJB、DCE、DCOM 等）与 CORBA 之间产生了迫切的可互操作性需求（可互操作性指在一个系统中用不同工具或不同供应商的产品开发的两个组件是否可以协同工作）。因此 CORBA 2.0 引入了通用 ORB 间协议 GIOP（General Inter-ORB Protocol）和基于 TCP/IP 协议的 Internet ORB 间协议 IIOP 来作为 ORB 间的通信协议标准，实现了不同供应商 ORB 产品间的可互操作性。

CORBA 通过平台独立性和语言无关性真正实现了跨平台，提供了一个允许在分布式和异构型环境中应用程序之间进行互操作的框架，大大促进了分布式计算技术的发展。

CORBA 参考模型也描绘了未来软件开发的方式：通用软件开发商开发组件，专业领域开发商提供框架，系统集成商开发应用。当网络条件和安全问题较好的解决后，软件将演变成一种服务，所有软件，组件、框架、应用，都挂接到 ORB 总线上，应用提供商选择合适的组件和框架搭建应用，提供了构件应用最好的灵活性；对最终用户而言，它们看到的只是构建好的应用也即服务。

1．2 面向服务的体系结构与 Web 服务

日益增长的电子商务和企业应用集成需求也促成了面向服务的体系结构(Service-Oriented Architecture，简称 SOA)的发展，SOA 不同于传统的分布式对象技术和软件组件技术，它更侧重于服务的发布、发现和使用。

Web Service 就是采用面向服务的体系架构的一种基于 Web 的新的分布式计算技术，它将现实世界中的业务活动抽象为服务，通过使用简单对象访问协议 SOAP（Simple Object Access Protocol)、通用描述/发现/集成 UDDI（Universal Description, Discovery, & Integration）和服务描述语言 WSDL（Web Services Definition Language）以 XML 文档形式进行信息交换。其组成如下：

Web 服务的核心组件及其层次

由于 Web Service 建立在现有的标准 Web 协议基础上，可以兼容现有的防火墙技术，并通过自描述的 XML 进行信息交换，简单通用，近几年得到了迅速发展和应用。

2．系统集成现状与需求

当前，由于独立封闭或使用多种不同技术实现的系统需要跟外部系统进行数据交换或交互的需要，系统集成变成了一个热点。

另外，随着分布式计算技术的发展，当前有许多可用的 ORB 产品，如商品化的 IBM Component Broker, IONA Orbix, Borland/Inprise VisiBroker 和开源的 Tao ORB，OmniORB，ORBit ORB，JacORB 等。其中 JacORB 是使用纯 JAVA 按照 ORB 标准实现的高性能多线程对象请求代理，当前版本是 2.3.0 版，是使用通用公共许可证 GPL 授权的，它提供了 IDL 编译器，支持 OMG IDL/JAVA 语言映射版本 2.3，支持 IIOP,GIOP 1.2,　　而且 JacORB 包含 IIOP 代理，也叫小程序代理。小程序代理可以处理端口限制，而通过 Http 隧道将 GIOP 包封装成 HTTP 包可以通过防火墙，充分利用现有的防火墙技术，大大提高了 JacORB 的安全性，使 JacORB 得到了广泛的应用。正是由于 JacORB 的这个特点，使用除了 Java 外的其他语言如 C++/C#等来访问 JacORB 时大大增加了困难。

那么，如何在一个系统中访问外部系统中通过 JacORB 提供的服务呢？

3．集成方案

为了解决这个问题，可以使用相应编程语言实现 IIOP 代理，但是这样将会花费大量时间，而且通用性将大打折扣，本文就此提出了一个解决方法：增加一个中间层，它同时作为 CORBA 的客户机和 Web Service 的提供者，使用 Java 语言访问 JacORB 提供的对象，将 CORBA 服务发布为 WebService，这样虽然性能有所损耗，但是更加简单通用。

3．1 访问 JacORB 服务

假定使用 JacORB 实现的 CORBA 服务器提供了以下 Demo.IDL 文件定义的服务：

```
module demo
{
    struct PatientInfo
    {
        string patientId;
        string name;
        string sex;
    };

    interface Query
    {
        long GetPatient(in string patientId,out PatientInfo one);
    };
  };
```

使用 JAVA 实现的 Client 端来调用服务器提供的服务很简单。如下代码所示：

```
Query query = null;
org.omg.CORBA.ORB orb = null;
try {
    orb = org.omg.CORBA.ORB.init(null,null);
    NamingContextExt nc = NamingContextExtHelper.narrow(
            orb.resolve_initial_references("NameService"));
    query = QueryHelper.narrow( nc.resolve(nc.to_name("query.demo")));
    PatientInfoHolder patient = new PatientInfoHolder();
    int result = query.GetPatient("007", patient);
    if (result == 0)    {
        PatientInfo info = patient.value;
        System.out.println(info.patientId  +  "\t"  +  info.name  +  "\t"  +
info.sex);
    }
}
catch (Exception e) {
    e.printStackTrace();
}
finally{
    if (orb != null) orb.shutdown(true);
}
```

3. 2 实现 Web Service

如何通过JAVA实现一个Web Service 呢？为简单起见，可以使用apache Axis来实现，它提供了创建服务器端、客户端和网关SOAP操作的基本框架。Axis目前版本是为Java编写的，不过为C++的版本正在开发中。

为了描述Web Service调用结果，先定义ServiceResult类：

```
package webservice;
public class ServiceResult
{
    private int code;       //调用 WebSercie 成功返回 0，失败返回 1，异常返回-1
    private String text;    //出错或发生异常时返回出错信息

    public ServiceResult(int c, String t)
    {
        this.code = c;
        this.text = t;
    }

    public int getCode(){ return this.code; }
    public void setCode(int target){ this.code = target; }

    public String getText(){ return this.text; }
    public void setText(String target){ this.text = target; }
};
```

为了返回查询结果，还需要定义 QueryResult 类：

```
package webservice;
import demo.*;

public class QueryResult
{
    private ServiceResult result;       // 操作是否成功
    private PatientInfo patientInfo;    // Corba 返回对象

    public QueryResult(ServiceResult r, PatientInfo info)
    {
        this.result = r;
        this.patientInfo = info;
    }

    public ServiceResult getResult(){ return this.result; }
    public void setResult(ServiceResult target) { this.result = target; }

    public PatientInfo getPatientInfo() { return this.patientInfo; }
    public void setPatientInfo(PatientInfo info){ this.patientInfo = info; }
};
```

然后实现 Web Service 如下：

```
package webservice;
```

```
import java.io.*;
import demo.*;

public class QueryService {
    public QueryResult GetPatient(String id) {
        QueryResult queryResult = null;
        Query query = null;
        org.omg.CORBA.ORB orb = null;

        try {
            // 初始化 ORB
            orb = org.omg.CORBA.ORB.init((String[])null,null);
            // 获取命名服务
        org.omg.CORBA.Object obj = orb.resolve_initial_references("NameService");
            NamingContextExt nc = NamingContextExtHelper.narrow(obj);

            // 解析对象
            query = QueryHelper.narrow(nc.resolve(nc.to_name("query.demo")));

            //使用对象
            PatientInfoHolder patient = new PatientInfoHolder();
            int result = query.GetPatient(id, patient);

            if (result == 0) {
                queryResult    =    new    QueryResult(new    ServiceResult(0,
"",patient.value);
            }
            else    {
                queryResult = new QueryResult(new ServiceResult(1, result + " 查
询失败！"), null);
            }
        }
        catch (Exception e) {
            queryResult = new QueryResult(new ServiceResult(-1, e.getMessage()),
null);
        }
        finally {
            //关闭 ORB
            if (orb != null) orb.shutdown(false);
        }

        return queryResult;
    }
```

3.3 发布 Web Service

然后编写 wsdd 文件进行服务发布，注意如果要返回一个复杂对象，一定要把返回结果涉及到的所有类都要进行进行定义，否则 Marshal/Unmarshal 失败。Query.wsdd 内容如下：

```
<deployment xmlns="http://xml.apache.org/axis/wsdd/"
xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">
<service name="QueryService" provider="java:RPC">
<parameter name="className" value="webservice.QueryService"/>
<parameter name="allowedMethods" value="*"/>

<beanMapping languageSpecificType="java:webservice.ServiceResult"
qname="ns1:ServiceResult" xmlns:ns1="urn:BeanService"/>
<beanMapping languageSpecificType="java:demo.PatientInfo"
qname="ns1:PatientInfo" xmlns:ns1="urn:BeanService"/>
<beanMapping languageSpecificType="java:webservice.QueryResult"
qname="ns1:QueryResult" xmlns:ns1="urn:BeanService"/>
</service>
</deployment>
```

3.4 测试与使用服务

配置好 Web 服务器、axis 和 JacORB 后，使用 axis 提供的工具将 Query.wsdd 定义 WebService 发布即可。

可使用 Internet Explorer 进行 Web Service 测试，使用 http://webserver:8080/axis/services/QueryService?wsdl 获得 Web Service 的服务定义，使用 http://webserver:8080/axis/services/QueryService?method=GetPatient&ind0=007 调用GetPatient服务来查询”007”的资料，将可获得对象Mashal后所得的XML表示，在程序中Unmashal即可得到对象，如果成功，由该对象可获得查询所得信息；如果失败，也可查询到详细的出错信息：Corba服务器出错、Web Service 服务器异常或者网络导致异常等等。

4. 结论

JacORB 产品的产品独有的特性是优势，但同时也增加了互操作性难度，在进行系统的集成时要使用成熟的新技术，同时更重要的是要充分利用遗留系统资源。本文所进行对此进行了探讨，通过增加中间层方法，充分利用 SOA 的优势，使系统集成走向灵活、通用，最大限度的提高系统的适应性。

在实现中有些细节需要特别注意，例如在 JAVA 中字符串跟其他编程语言字符串存储与编码方式的不同、服务发布和使用、应用服务器使用等。

另外，由于涉及到网络信息交换、Mashal/Unmashal 操作、某些编程语言自动生成 Web Service

引用代理类问题等，可以使用网络监视工具截获网络包帮助分析，从而发现问题所在，axis 提供的 tcpmon 工具就很有用。

由于增加了中间件，性能有所损耗；安全性也是一个考虑；另外硬件资源也有所要求和增加。需要按实际需要选择。

参考资料：

1. Bruce Eckel 著 陈吴鹏等译《Java 编程思想》第三版 机械工业出版社

2. 李文军等《分布式对象技术》机械工业出版社

3, Eric Newcomer 等著 徐涵译《Understanding SOA with Web Services 中文版》电子工业出版社