Java 到 IDL 映射方法的分析和改进

Improvement and Analysis of the Method of Mapping JAVA to IDL

姚世军　张 平　王宏勇　马国强
（解放军信息工程大学理学院 郑州经济管理干部学院 测绘学院 450052）

1 引言

EJB 规范和 CORBA 规范是目前三层或多层结构的主要组件开发结构。EJB 组件是基于 JAVA 的组件规范，而 CORBA 是与语言无关的组件规范。在一个大的复杂的应用系统中，通常可能会有这两种组件同时存在，因此它们之间的互操作是人们必须考虑的问题。这种互操作包括 JAVA 客户调用 CORBA 对象或非 JAVA 客户调用 EJB 对象，以及 EJB 对象与 CORBA 对象之间的互操作。

EJB 规范中包括关于与 CORBA 的互操作部分。它详细说明了命名服务、事务服务和安全服务等方面的透明互操作。如果应用服务器支持 CORBA 互操作，CORBA 可以直接通过 IDL 接口来调用 EJB。任何 JAVA 代码也都可以通过 JAVA ORB 来调用 CORBA 对象。在两个独立开发的 EJB 和 CORBA 系统要集成到一起时，EJB 组件可以调用 CORBA 服务，同时 CORBA 客户也要使用 EJB 系统中的关键服务。

CORBA 使用 IDL 语言来定义客户与调用对象之间的接口，而 EJB 是用 JAVA 来定义它的所有接口，为了实现 CORBA 客户对 EJB 组件的调用，必须将 JAVA 接口转换成 IDL 接口定义，即完成 JAVA－IDL 的映射。但是这种由 JAVA 到 IDL 映射生成的 IDL 接口对 CORBA 客户来说并不好用，并且对 EJB 设计有特殊要求。

本文将重点讨论标准的 JAVA－IDL 映射互操作方法和这种方法中存在的问题，最后提出解决这些问题的改进方法。

2 JAVA－IDL 直接映射方法

CORBA－EJB 互操作的规范中定义了四种映射：把 EJB 接口映射成 CORBA 接口；将 JNDI 命名方式映射成 CosNaming 命名方式；在 CORBA 环境和 EJB 环境中传播事务；在 CORBA 和 EJB 环境之间安全的传播各种信息。

后三种映射是非常重要的，但如果没有应用服务器来实现无逢的安全和事务处理，这些实现是很困难的。从开发者的观点它们是不可见的。因此开发者最感兴趣的问题是分布式对象之间映射，即 EJB 接口到 CORBA 接口的映射。

在完全支持 EJB－CORBA 映射的应用服务器中，EJB 对象可以自动成为 CORBA 对象。IDL 接口通常是隐含的从 EJB 远程 JAVA 接口中生成。JAVA 命名服务 JNDI 可以映射成 CORBA 命名服务，这样 EJB HOME 对象可以出现在 CORBA 命名服务中。CORBA 客户可以直接在 CORBA 命名服务中查询 EJB Home 对象，并把 EJB 对象用作 CORBA 对象，不需要其他额外工作。完成 JAVA－IDL 映射的具体步骤如下：

(1) 根据 EJB 远程接口使用 RMI 编译程序生成 IDL (rmic－idl －noValueMethods)。

(2) 使用 CORBA IDL 编译程序生成客户编程语言的存根程序，并将其编译到客户程序。

(3) 配置 EJB 服务器使其将 CORBA CosNaming 服务作为它的 JNDI 提供者。

(4) CORBA 客户在 CORBA 命名服务中找到 EJB 对象。

(5) CORBA 客户对 EJB 对象进行调用。

下面定义一个 EJB 对象，并给出了 Java－IDL 之间的映射。

```
import java.util.*;
import javax.ejb.EJBObject;
import java.rmi.RemoteException;
public interface cart extends EJBObject;
{ public void addBook(String title) throws
BookException,RemoteException;
public Vector getContents() throws RemoteEx-
ception;}
//标准的 JAVA－IDL 的映射
interface Cart: :Javax::ejb::EJBObject
{
void addBook(
in :: CORBA ::WStringValue argo);
void removeBook(
in ::CORBA::WStringValue argo) raises(
BookEx);
readonly attribute :: java:: util:: Vector con-
tents;
};
```

3 JAVA－IDL 映射中的问题

虽然 JAVA－IDL 直接映射在理论上是可行的，但是这种映射生成的 IDL 接口存在许多问题，很难在实际中应用。

3.1 映射目标的冲突

JAVA－IDL 映射有两个目标，一个要求支持 Java 的 RMI/IIOP：即允许 JAVA 通过 CORBA 的 IIOP 进行远程方法调用；另一个目标是支持 CORBA 客户，即要使 CORBA 客户可以使用 JAVA 远程对象。第一点要求从 JAVA 转换成 IDL 不能丢失任何信息，而这只有通过 IIOP 的 JAVA－JAVA 通信才有可能实现。第二个目标又要求映射易于使用，但这样目标 1 的要求变成次要的。为了避免丢失信息，映射中要将 JAVA 细节显示在 IDL 中，而这些对非 JAVA 客户没有用，且在多数情况会带来严重甚至是至命的错误。

3.2 值类型的问题

CORBA 在 IDL 中引入值类型，以按值对象传递。值类型有公共和私有数据成员、操作、初始化程序和继承。与 IDL 接口不同的是，值类型不表示远程对象，它的实例是按值传递，即传递数据成员并在另一端建立备份。如果值类型有操作，接收者必须实现每个操作。CORBA 中值类型无处不在，但有许多 ORB 不支持它们；同时值类型增加了 IDL 的复杂性。多个平台互操作的开发者都希望避开它们。

3.3 集合问题

JAVA 程序员通常使用对象集合，因为 JAVA 没有定义强类型的集合特性。映射这类的集合可能产生弱类型的接口，使客户端很难使用。

3.4 其他问题

会有太多的 JAVA 类被用在 IDL 中，包括 JAVA 类的私有数据以及完全与非 JAVA 客户无关的继承类中的部分。

直接映射会在深的嵌套目录结构中生成许多文件，这对 JAVA 语言来说是自然的，但对其他语言来说就不是这样了。

由于 CORBA 通常是以非层次的管理空间，一个模块一个文件，这样 JAVA 包被映射成不同文件中的多个 IDL 模块并重新打开。虽然技术上可以在多个文件中复杂打开一个模块，但在有些语言中可能带来问题。

异常不匹配。JAVA 允许异常继承，但 IDL 不允许这样。为了保持这种继承，JAVA 异常被映射成 CORBA 中异常的值类型，而这对 CORBA 程序员来说是非常不正常的算法。同时对允许异常继承的其他语言也没有用。

4 Java－IDL 映射方法的改进措施

为了使 CORBA 客户容易使用 EJB 组件，在设计 EJB 组件时可以采取多种措施将直接映射中出现的问题最小化。

4.1 用 IDL 来定义 EJB 数据类型

使 EJB 对 CORBA 客户更容易使用的技术之一是用 CORBA 的 IDL 来定义 EJB 中使用的数据类型。IDL－JAVA 的映射是开发基于 JAVA 的 CORBA 系统的基础，它易于使用，因为设计 IDL 的目的就是为了映射成象 JAVA 这样的语言，但反之是不对的。这就意味着任何 IDL 类型，在 IDL－JAVA 映射中都有对应的派生的 JAVA 类型。Java－IDL 映射是这些派生类型的特例。EJB 参数(从 IDL 派生的 JAVA 类)可以映射成包含原 IDL 类型的值 BOX。例如：

```
//IDL 的数据类型定义
struct Customer {
string name;
string address;}
//EJB:使用 IDL 派生的 Customer 类的接口
interface Mailer extends EJBObject{
void sendMail (Customer customer);}
// IDL :Java－IDL 映射结果
interface Mailer: ::javax::ejb::EJBObject{
void sendMail(
in ::org::omg::boxedIDL::Customer customer
);}
```

::org ::omg::boxedIDL::Customer 类型是值盒子：即包裹原 IDL 定义的 Customer 类型的实例。假设客户端的 ORB 支持值类型，对客户端编码来说可以把它当成 IDL 类型一样用。

4.2 根据 IDL 来生成 EJB

对于大多数 EJB 接口可以完全用 IDL 来定义 EJB 系统中的远程接口，然后使用这些 IDL 接口到 JAVA 的映射派生出 EJB 接口。这样生成的 EJB 接口将自然遵守 IDL 友好的规则。此时应考虑到 JAVA 仅支持 IN 类型的参数传递。为了支持 IDL 的 out 和 inout 参数，IDL－JAVA 映射时在 JAVA 中引入了 Holder 对象。JAVA 的 ORB 使用 Holder 来与调用者用 out 或 inout 参数返回斩值进行通信。

根据 IDL 生成 EJB 的具体步骤如下：

(1) 使用 IDL 定义接口，并且仅用 In 参数传递模式，避免 valuetype。

(2) 根据 IDL 派生 EJB 接口。对由 IDL－JAVA 映射生成的结果进行修改。即添加 extends javax. ejb. EJBObject 和到该接口中，添加 throws java. rmi. RemoteException 到每个操作中。可以使用代码生成工具使这个过程自动化。

(3) 为第 2 步中建立的 EJB 接口编写 EJB 代码。

(4) 根据第 2 步中的 EJB 接口编写客户程序。

(5) 部署 EJB 和直接通过 RMI/IIOP 与 EJB 通信的客户。

4.3 其他方法

(下转第 54 页)

(上接第 66 页)

为了 CORBA 客户更方便地使用 EJB 对象,还可以使用下面的改变方法:

(1) 用 IDL 来定义异常。这类似于上面用 IDL 来定义数据类型。在 EJB 的 THROW 子句中派生的 IDL 异常仅映射回它原来的定义,当在 JAVA－IDL 映射时。这是避免映射中产生异常值类型的唯一方法。

(2) 使用数组,而不使用集合类。Java 数组被映射成 IDL 的 VALUE－BOX,它包括一组 IDL 相应类型。这种方法要优于 JAVA 值类型的<any>弱类型。

(3) 使用公共数据类型。如果使用非 IDL Java 类作为数据类型,要使重要成员公共化,这样会使非方法映射更易于使用。

(4) 从映射的观点,使用 Java. lang. object 是有好处的。这可以映射成 IDL 的 any 类型,它是自解释的数据,CORBA 客户可以在运行时解释它。

(5)避免套管现象的出现。使用无值对象方法映射可以避免把无用的 JAVA 类引进到映射所生成的 IDL 中。建议不要使用带方法映射。即使在最简单的测试情况下,它生成的 IDL 也是太复杂。在现实中它几乎不能工作。

以上技术措施可能帮助 CORBA 客户来访问 EJB,从而使 CORBA 与 EJB 的互操作更加容易实现和使用。

参考文献

1 IONA Technologies. IONA E2A Web Services Integration Platform.

2 Mastering Enterprise JavaBeans (Second Editions), ED Roman, 刘晓华等译。

3 Oracle9iAS J2EE 应用程序开发, Nirva Morisseau－Leroy, 周立斌等译。