RMI trên IIOP (RMI-OVER-IIOP)

Bởi:

Khoa CNTT ĐHSP KT Hưng Yên

Một số khái niệm cơ bản

Common Object Request Broker Architecture (Corba) là đặc tả của Object Management Group - OMG giành cho việc đạt được tính tương tác giữa các nút tính toán phân tán. Mục tiêu của chúng đã định nghĩa một cấu trúc mà có thể cho phép những môi trường không đồng nhất có thể liên lạc tại mức đối tượng, không quan tâm đến người nào đã thiết kế ra hai điểm cuối của ứng dụng phân tán. Corba là một ngôn ngữ trung lập được thực thi rộng rãi hơn trên các nền platform khác nhau hơn là COM. Nhưng có vài tính không tương thích giữa những thực thi của các nhà cung cấp khác nhau, Corba sử dụng một giao thức gọi là IIOP (Internet Inter-ORB Protocol) để liên lạc giữa các hệ thống khác nhau.

Triệu gọi phương thức từ xa (RMI) và kiến trúc môi giới yêu cầu đối tượng chung (Corba) là tầng giữa hỗ trợ việc gọi phương thức phân tán. Tuy nhiên RMI trong Java có nhiều điểm khác và và hạn chế so với Corba.

Sự khác nhau của Corba và RMI trong Java thể hiện ở các điểm sau:

- RMI là một phần của bộ J2SDK và là hàm thư viện hỗ trợ các lời gọi phương thức từ xa và trả về giá trị cho các ứng dụng tính toán phân tán. Chúng ta giả sử rằng ngôn ngữ Java được sử dụng ở cả hai phía gọi và phía bên phương thức được gọi (hay gọi chung là ngôn ngữ thuần Java).

- Corba là một chuẩn công nghiệp cho phép gọi các phương thức từ xa và nhận kết quả trả về nhưng không giống như RMI mà nó có thể được sử dụng khi bên phía gọi và bên phía phương thức được gọi có thế sử dụng các ngôn ngữ lập trình khác nhau, bao gồm cả trường hợp là cả hai bên đều không sử dụng ngôn ngữ java.

- RMI là một tập hàm các thư viện đơn giản vì cả hai bên đều sử dụng cùng một ngôn ngữ lập trình và kiến trúc máy. Điều này sẽ làm cho vấn đề triệu gọi phương thức từ xa dễ giải quyết hơn.

- Như đã đề cập ở trước thì Corba định nghĩa nhiều dịch vụ, một trong những dịch vụ đó thực hiện chức năng tương tự như RMI, nhưng Corba thì hoạt động với nhiều ngôn ngữ lập trình khác nhau và không chỉ với nền java.

- Nền của Java chứa một Corba ORB, Corba dùng cùng kỹ thuật Stub/Skeleton như RMI nhưng không giống với RMI mà Corba phát sinh stub và skeleton từ một mô tả giao diện độc lập với ngôn ngữ được gọi là ngôn ngữ "Mô tả giao diện" (Interfaca Description Language- IDL) thay vì mã nguồn của ngôn ngữ IDL xác định tên phương thức, cũng như tham số gọi và trả về theo một kiểu ngôn ngữ trung lập.

+ RMI của Java lập trình đơn giản và dễ hiểu hơn Corba. Các đối tượng RMI sử dụng giao thức JRMP (Java Remote Method Protocol) để "nói chuyện" với nhau. Trong khi đó các đối tượng Corba lại sử dụng giao thức IIOP được chấp nhận và chuẩn hoá của hầu hết các đối tượng lớn.

Tại sao lại phải sử dụng RMI trên IIOP

Như trên chúng ta đã biết qua các khái niệm cơ bản về IIOP và RMI, sự khác nhau cơ bản giữa RMI với Corba….Nhưng nhìn chung RMI và Corba chưa có tiếng nói chung đồng nhất. Vì lý do này mà Java cung cấp cho ta cách cài đặt để các đối tượng RMI của Java có thể sử dụng giao thức IIOP để giao tiếp với các đối tượng của Corba. Khả năng này được gọi là RMI trên IIOP (RMI-Over IIOP). Một kỹ thuật RMI với cách cài đặt RMI-Over-IIOP sẽ vừa có thể giao tiếp được với các trình khách RMI vừa có thể giao tiếp dược với các đối tượng Corba.

Từ sơ đồ hình 6 ta thấy:

-Nếu trình khách viết bằng Java thì:

+ Có thể giao tiếp được với đối tượng chủ RMI theo giao thức thuần Java (JRMI) hoặc giao thức RMI-Over-IIOP

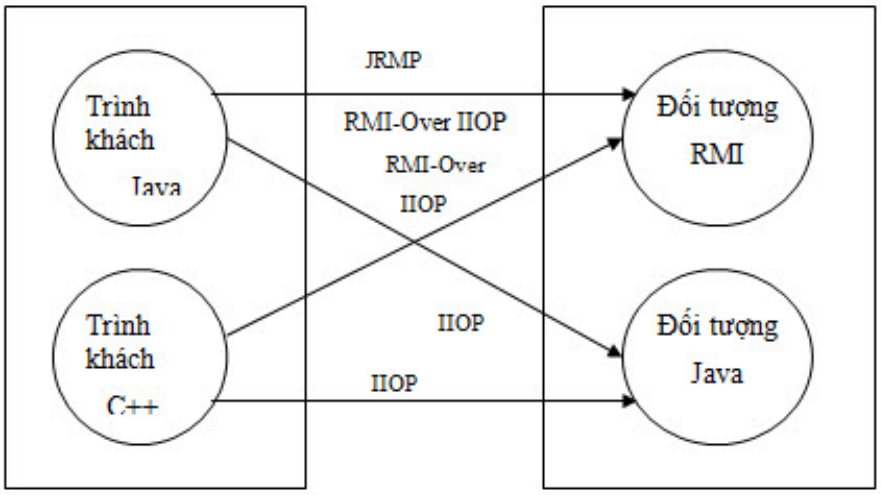
+Hay có thế giao tiếp được với đối tượng Corba theo giao thức IIOP

- Nếu trình khách viết bằng C++ thì:

+Có thể dùng giao thức IIOP để triệu gọi đối tượng Corba

+Hoặc chỉ có thể giao tiếp được với đối tượng RMI thông qua giao thức RMI-Over-IIOP mà thôi.

- Cả hai đối tượng RMI và Corba có thể giao tiếp được với nhau dựa trên giao thức RMI-Over-IIOP.



Mô tả các khả năng giao tiếp bằng giao thức RMI-Ovẻ IIOP so với các giao thức JRMP và IIOP giữa các đối tượng.

Xây dựng đối tượng RMI giao tiếp bằng IIOP

Với RMI-IIOP, stub và skeleton được phát sinh ngay từ định nghĩa đối tượng java. Thay vì sử dụng giao thức trong RMI để liên lạc giữa 2 tiến trình, RMI-IIOP sử dụng giao thức Corba IIOP để nó có thể gọi các đối tượng không được viết bằng ngôn ngữ java. Các bước phát triển bắt đầu với cài đặt một lớp phát sinh bằng IDL trong ngôn ngữ java. Lớp này được biên dịch với trình tiện ích rmic dùng cờ hiệu –iiop –d và một lớp bên phía máy chủ sử dụng COSNaming, truy cập JNDI (Java Naming and Directory Interface – JNDI).

Cài đặt và thực thi RMI-Over-IIOP

Các bước để tạo và biên dịch một chương trình RMI-Over-IIOP

Bước 1: Đặc tả giao tiếp Interface cho đối tượng RMIOjects

Ví dụ: RMIObject.Java

```
package rmi;

import java.rmi.*;

public interface RMIObject extends Remote {

public String tellAbout () throws RemoteException;

}
```

Bước 2: Cài đặt đối tượng RMIObject

Ví dụ: RMIObjectIpml.Java

```
package rmi;

import java.rmi.*;

public class RMIObjectImp1 implements RMIObject{

public String tellAboult() throws RemoteException{

System.out.print(“client asking”);

Return “ Toi la RMI day”;

}

}

}
```

Đối tượng của ta cung cấp một phương thức tellAboult () để “xưng danh” với trình khách. Như vậy bạn đã có một đối tượng RMI thuần nhất viết toàn bằng ngôn ngữ java. Bạn biên dịch đoạn chương trình này như sau:

```
javac *.java;
```

Với lớp đối tượng RMIObjectImpl.class thu được ta muốn đối tượng phải có khả năng giao tiếp bằng giao tiếp IIOP thay cho giao thức JRMI mà Java quy định đối với RMI. Điều này có thể thực hiện dễ dàng bởi trình dịch rmic.exe với thông số -iipp như sau

```
rmic.exe –iiop RMIObjectImpl
```

Kết quả bạn sẽ thu được hai tập tin trung gian là: RMIObjecStub.Java và RMIObjectTie.java. Đây là hai lớp chịu trách nhiệm chuyển đổi lời gọi giao thức JRMI thành giao thức IIOP và ngược lại.

Trình cài đặt cho đối tượng RMI “giả” đối tượng Corba được viết như sau:

Ví dụ :

```
package rmi;
```

```
import javax.rmi.*;

import javax.naming.*;

public class Setup{

public static void main(String [] args) throws Exception{

//Tạo thể hiện của đối tượng Java

RMIObject obj=new RMIObject();

// yeu cau may ao Java nhan dang obj la mot doi tuong cua Corba

PortableRemoteObject.exportObject(obj);

System.out.println("Binding object…..");

// lay ve dich vu Context cua dih vu dang ky ten

Context ctx=new InitialContext();

/* Rang buoc ten cua doi tuong voi dich vu quan ly tin.Sau loi goi bind()

Chuong rinh se roi vao trang thai cho cac y/cau tu may khach

*/

ctx.bind("My RMI as Corba",obj);

System.out.println("client yeu cau cho doi ");

}

}
```

Khi viết chương trình cài đặt,có một số điểm mà bạn cần chú ý như sau:

Để chuyển đối tượng RMI thành đối tượng có khả năng sử dụng giao thức IIOP chúng ta cần:

1. Thay thư viện *import java.naming*;*

import java.rmi.;*

bằng thư viện:

import javax.naming.;*

import javax.rmi.;*

2. Thay vì gọi *UniscastRemoteObject.exportObject()* để thông báo cho máy ảo Java biết sự hiện diện của đối tượng RMI thì chúng ta gọi:

PortableRemoteObject.exportObject(obj);

Lệnh trên sẽ thông báo tới máy ảo Java biết rằng đối tượng obj có thể giao tiếp từ xa bằng giao thức IIOP của Corba thay cho JRMI trong RMI.

Các đối tượng Corba cần đăng ký với dịch vụ quản lý tên tnameserv (COS naming) trước khi bị truy xuất bởi trình khách. Trong Corba để đăng ký dịch vụ COS Naming thì bạn gọi lệnh đăng ký sau:

org.omg.Corba.Object nameService =orb.resolve_initial_references ("NameService");

NamingContext nsContext=NamingContextHelper.narrow (nameService);

Thay cho đoạn lệnh truy tìm context gốc của dich vụ COS Naming trên thư viện javax.naming cung cấp cho bạn đối tượng tổng quát và gọn hơn để lấy về context gốc như sau:

Context ctx=new InitialContext ();

Với đối tượng context gốc có được ,chúng ta ràng buộc RMIObject với dịch vụ COS Naming như sau:

ctx.bind ("My RMI as Corba", obj);

Để chỉ định đối tượng context gốc lấy về là đối tượng của dịch vụ COS Naming bạn chạy trình Setup từ dòng lệnh sau:

java -Djava.naming.factory.initial=com.sun.jndi.cosnaming.CNCtxFactory -Djava.naming.provider.url=iiop://localhost:900

- Chú ý:

1.Java sẽ sử dụng dịch vụ lớp khởi tạo com.sun.CNCtxFactory làm dịch vụ đăng ký tên

2.Tham số java.naming.provider.url chỉ định vị trí mà dịch vụ quản lý tên đang sử dụng (trước khi chạy trình Setup bạn nhớ khởi động tnameServerv ở một cửa sổ DOS-Prompt khác với cổng mặc định của tnameserverv là 900).

Như vậy là trình Setup đã biên dịch thành công đối tượng RMIObject có khả năng giao tiếp được với trình khách bằng giao thức IIOP của Corba.

Bước 3:Viết trình khách truy xuất đối tượng RMI theo giao thức IIOP:

Trình khách được viết theo mô hình Corba thay cho RMI.Vì vậy đối tượng chủ phải có khả năng cung cấp cho trình khách tập tin .idl để trình khách có thể tạo ra các lớp trung gian. Và theo đó trình dịch rmic.exe sẽ giúp chuyển interface của Java sang đặc tả .idl của Corba. Chúng ta sẽ gọi chương trình dịch rmi.exe với tham số dòng lệnh .idl và yêu cầu rmic.exe diễn dịch lớp RMIObject từ Java sang IDL như sau: rmic.exe –idl RMIObject

Chú ý: Ở các phiên bản cũ của Java bạn có thể sử dụng trình diễn dịch Java2idl.exe thay cho lệnh .idl.

Sau khi trình biên dịch chạy xong ta thu được tập tin RMIObjec.idl như VD sau:

Ví dụ : RMIObject.idl

```
/* RMIObject.idl

Generated by rmic-idl.Do not edit

Friday,Ju 7, 2008 8:30:00 PM PDT

*/

#include “orb.idl”

#ifndef__RMIObject__

#define__RMIObject__

Interface RMIObject{

::Corba::WstringValue tellAbout();

};

#pragma ID RMIObject “RMI:RMIObject:0000000000000000”
```

#endif

Ta thấy sau khi trình biên dịch thực hiện xong ta có phương thức tellAbout() trả về kiểu chuỗi viết bằng ngôn ngữ của Corba. Chúng ta sẽ gọi trình jidl để sinh ra các tập tin trung gian với trình khách theo cách sau:

idlj –fclient RMIObject.idl

Và sau đây mà chương trình demo cài đặt cho trình khách:

Ví dụ : Client.Java

```
import org.omg.Corba.*;

import org.omg.CosNaming.*;

public class Client{

public static void main(String [] args) throws Exception{

//khoi dong trinh moi gioi trung gian

ORB orb=ORB.init (args,null);

//tim tham chieu cua nut context goc

org.omg.Corba.Object nameService=

orb.resolve_initial_references (“NameService”);

//lay ve tham chieu cua doi tuong

NamingContext nsContext=

NamingContextHelper.narrow (nameService);

NameComponent c=new NameComponent (“My RMI as Corba”,” ”);

Namecomponent path[]={nc};

RMIObjectservant= RMIObjectHelper.narrow

(nsContext.resolve(path));
```

```
//Trieu goi doi tuong

System.out.println ( servant.tellAbout() );

}

}
```

Biên dịch và chạy chương trình khách như sau:

Biên dịch:

javac *.java

Chạy chương trình:

java Client

Và kết quả thu được trên màn hình là

Toi la RMI day

Chương trình Client.Java trên đây hoàn toàn là một trình khách Corba bình thường.

Ngoài ra chúng ta có thể sử dụng đối tượng Context để viết trình khách theo phong cách RMI như ví dụ 6 dưới đây:

Ví dụ : ClientRMI.Java

```
import Javax.rmi.*;

import Javax.naming.*;

public class ClientRMI {

public static void main (String [] args) throws Exception {

Context cxt=new InitialContext ();

RMIObject servant = (RMIObject) PortableRemoteObject.narrow (

Cxt.lookup (“ My RMI as Corba ”), RMIObject.class );

System.out.println (servant.tellAbout() );
```

}

}

Và cũng tương tự như trình chủ để chạy được trình khách viết theo phong cách RMI truy tìm được dịch vụ COS Naming ta phải gọi máy ảo Java với tham số dòng lệnh tương tự như sau:

-Djava.naming.factory.initial=

com.sun.jndi.cosnaming.CNCtxFactory

-Djava.naming.provider.url=iiop://localhost:900

Và chúng ta cũng thu được kết quả tương tự như trên ví dụ 5 là

Toi la RMI day

Khởi tạo các thông số ngữ cảnh

Khi xây dựng chương trình khách và chỉ dùng đến đối tượng Context như ví dụ 3 và ví dụ 5, chúng ta đã phải truyền đối số từ dòng lệnh để khởi tạo dịch vụ java.naming.factory.initial khá dài và phức tạp. Thay vì truyền các đối tượng Context cho dòng lệnh chúng ta có thể thực hiện ngay các thiết lập này ngay trong mã lệnh. Điều này có thể làm dễ dàng như trong ví dụ dưới đây:

Ví dụ : ClientRMIConfig.Java

import java.until.*;

import Javax.naming.*;

import Javax.rmi.*;

public class ClientRMIconfig{

public static void main(String [] args)throws Exception {

//phan thiet lap khoi tao cho context ngay trong ma lenh

Hashtable hat=new Hashtable();

hat.put (“ java.naming.corba.applet”,this);

```
hat.put(“java.naming.factory.initial”,

”com.sun.jndi.cosnaming.CNCtxFactory”);

hat.put(“java.provider.url”,”iiop://localhost:900”);

Context cxt=new Initialcontext (hat);

RMIObject servant=(RMIObject) PortableRemoteObject.narrow

(cxt.lookup(“My RMI as Corba”),RMIObject.class);

System.out.println(servant.tellAbout() );

}

}
```

Như bạn đã thấy chúng ta đã sử dụng bảng hat kiểu Hashtable để lưu các giá trị thiết lập cho Context. Thay cho lời gọi khởi tạo InitialContext() với giá trị rỗng,đối tượng Context được gọi khởi động với thông số thiết lập từ bảng hat như sau:

```
Context cxt=new Initialcontext(hat);
```

Bằng cách này trình khách có thể chỉ cần gọi đơn giản từ dòng lệnh :

java ClientRMIConfig