PHẦN III. LẬP TRÌNH PHÂN TÁN
CHƯƠNG IV
KỸ THUẬT LẬP TRÌNH PHÂN TÁN ĐỐI TƯỢNG RMI

I. GIỚI THIỆU LẬP TRÌNH PHÂN TÁN VÀ RMI

(Remote Method Invocation)

1. Giới thiệu kỹ thuật lập trình phân tán

Kỹ thuật lập trình phân tán thực chất là kỹ thuật lập trình phân tán mã lệnh hay đối tượng. Nó cho phép phân bố tải lên toàn mạng để tận dụng tài nguyên mạng giải quyết bài toán lớn, phức tạp thay vì tập trung trên cùng một máy. Các thành phần mã lệnh phân tán "kết cặp" với nhau một cách chặt chẽ, khác với lập trình socket là "kết cặp" lỏng lẻo. Một điểm khác cơ bản nữa của lập trình phân tán so với lập trình socket là: Socket là giao diện, còn các kỹ thuật lập trình phân tán như RPC, RMI...là cơ chế truyền thông.

Hiện này có nhiều kỹ thuật lập trình phân tán khác nhau như:

- Kỹ thuật gọi thủ tục từ xa RPC(Remote Procedure Call)
- Kỹ thuật gọi phương thức từ xa RMI(Remote Method Invocation)
- Kỹ thuật mô hình đối tượng thành phần phân tán DCOM
- Kỹ thuật kiến trúc môi giới trung gian CORBA
- Kỹ thuật EJB, WebService, RPC-XML...

Các kỹ thuật lập trình phân tán hiện này đều hướng đến mô hình đa tầng với kỹ thuật lập trình hướng dịch vụ(SOP) mà tiêu biểu là WebService. Vì nó cho phép giải quyết các bài toán lớn, phức tạp hiệu quả và nhiều ưu điểm khác. Kỹ thuật lập trình RMI tương tự như kỹ thuật RPC nhưng khác ở chỗ: Trong RPC chương trình client gọi thủ tục phía Server , còn trong RMI client gọi phương thức từ xa ở phía server(hướng đối tượng).

2. Giới thiệu kỹ thuật lập trình RMI

2.1. Đặc trưng của kỹ thuật RMI

RMI là kỹ thuật lập trình phân tán đối tượng, nó cho phép gọi phương thức của đối tượng từ xa qua mạng và nhận kết quả trả về từ máy từ xa.

RMI là một cơ chế truyền thông và là kỹ thuật thuần Java. Điều đó nghĩa là, kỹ thuật RMI chỉ cho phép các đối tượng thuần Java mới gọi từ xa phương thức của nhau được. Còn các đối tượng viết bằng ngôn ngữ khác như Delphi, C^{++}... thì kỹ thuật RMI không cho phép.

Chương trình ứng dụng phân tán RMI cũng đượ tổ chức theo mô hình client/server:

- Phía server là phía máy tính từ xa chứa các đối tượng có phương thức cho phép gọi từ xa.
- Phía client là phía chứa các đối tượng phát sinh lời gọi phương thức từ xa.

Một chương trình Client có thể kích hoạt các phương thức ở xa trên một hay nhiều Server. Tức là sự thực thi của chương trình được trải rộng trên nhiều máy tính. Đây chính là đặc điểm của các ứng dụng phân tán. Nói cách khác, RMI là cơ chế để xây dựng các ứng dụng phân tán dưới ngôn ngữ Java.

Mỗi đối tượng có phương thức cho phép gọi từ xa, trước khi sử dụng được nó phải được đăng ký với máy ảo java thông qua bộ đăng ký của JDK hoặc do người sử dụng định nghĩa. Và mỗi đối tượng đó cũng phải được gán một chuỗi dùng làm tên để truy xuất tìm đối tượng trên mạng. Chuỗi tên đó có dạng như URL:

"rmi://<host>[:port]/ObjName"

Trong đó:

- rmi : chỉ phương thức truy cập
- host: địa chỉ của máy trạm chứa đối tượng từ xa cần tìm
- port: Chỉ ra số cổng được sử dụng để truy xuất tìm đối tượng, nó có thể có hoặc không. Trong trường hợp không khai báo thì nó mặc định lấy số cổng 1099.
- ObjName: Là chuỗi tên gán cho đối tượng có phương thức cho phép gọi từ xa.

RMI sử dụng giao thức truyền thông JRMI. Giao thức này cho phép tạo ra môi trường mạng truyền thông trong suốt mà từ đó lời gọi phương thức từ xa không khác gì lời gọi cục bộ. Và để truyền thông, java sử dụng 2 đối tượng trung gian để đóng gói truyền thông và khôi phục lại lời gọi, kết quả thi hành phương thức từ xa qua mạng từ các gói tin truyền qua mạng đó.. Đối tượng _Skel cài phía bên server và _Stub cài phía bên client.

Để hỗ trợ lập trình RMI, java hỗ trợ nhiều lớp và giao diện thư viện mà tập trung chủ yếu trong 2 gói: java.rmi và java.rmi.server.

Như vậy kỹ thuật lập trình phân tán đối tượng RMI trong java đã cho phép phân tán tải lên các máy tính trên mạng thay vì tập trung trên một máy. Điều này thật sự có ý nghĩa lớn đối với các ứng dụng mà có khối lượng tính toán lớn mà đòi hỏi thời gian thực. Vì một máy tính có mạnh đến mấy cũng vẫn hữu hạn. Nhất là đối với những bài toán thực tế như: Bài toán thị trường chứng khoán, ngân hàng, bài toán hàng không, dự bào thời tiết, bài toán nghiên cứu vũ trụ...Phần sau đây chúng ta sẽ đi sâu vào nghiên cứu các kỹ thuật lập trình của RMI và cơ chế hoạt động của chúng.

2.1.. Kiến trúc của chương trình Client – Server theo cơ chế RMI

Hình 5.1. là kiến trúc của một chương trình phân tán đối tượng RMI theo mô hình Client /Server:

Trong đó:

- Server là chương trình cung cấp các đối tượng có thể được gọi từ xa.
- Client là chương trình có tham chiếu đến các phương thức của các đối tượng có phương thức cho phép gọi từ xa trên Server.
- Stub là đối tượng môi giới trung gian phía client.

- Skeleton là đối tượng môi giới trung gian cài phía server.
- Remote Reference Layer là lớp tham chiều từ xa của RMI.

Hình5.1.. Kiến trúc Client/Server của chương trình RMI

2.2. Các cơ chế hoạt động RMI

Trong một ứng dụng phân tán cần có các cơ chế sau:

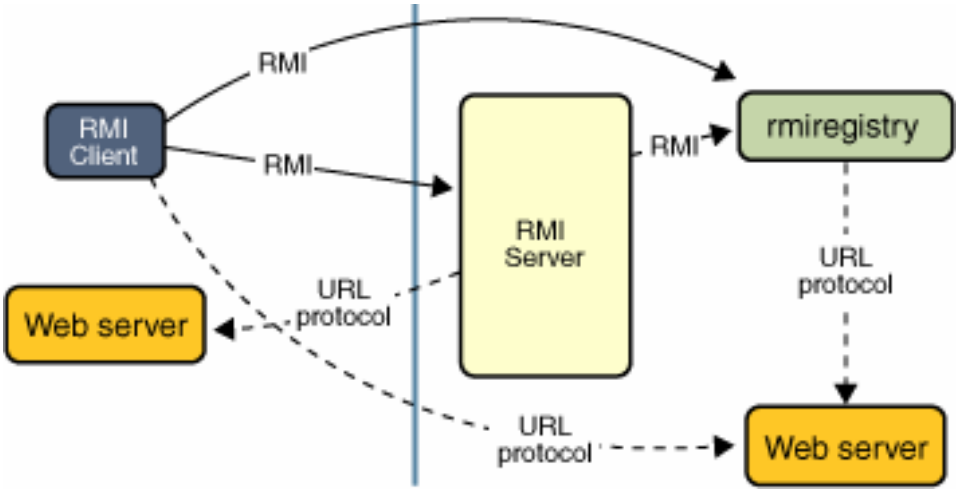
- Cơ chế định vị đối tượng ở xa
- Cơ chế giao tiếp với các đối tượng ở xa
- Tải các lớp dạng bytecodes cho các lớp mà nó được chuyển tải qua lại giữa JVM

1.3.1.Cơ chế định vị đối tượng ở xa (Locate remote objects): Cơ chế này xác định cách thức mà chương trình Client có thể lấy được **tham chiếu** (Stub) đến các đối tượng ở xa. Thông thường người ta sử dụng một dịch vụ danh bạ (Naming Service) lưu giữ các tham chiếu đến các đối tượng cho phép gọi từ xa mà client sau đó có thể tìm kiếm.

1.3.2. Cơ chế giao tiếp với các đối tượng ở xa (Communicate with remote objects): cơ chế truyền thông với các đối tượng từ xa được cài đặt chi tiết bởi hệ thống RMI.

1.3.3. Tải các lớp dạng bytecodes cho các lớp mà thực hiện chuyển tải qua lại giữa Máy ảo (Load class bytecodes for objects that are passed around): Vì RMI cho phép các chương trình gọi phương thức từ xa trao đổi các đối tượng với các phương thức từ xa dưới dạng các tham số hay giá trị trả về của phương thức, nên RMI cần có cơ chế cần thiết để tải mã Bytecodes của các đối tượng từ máy ảo này sang máy ảo khác.

Hình 5.2. mô tả một ứng dụng phân tán RMI sử dụng dịch vụ danh bạ để lấy các tham chiếu tới các đối tượng ở xa.



Hình 5.2. Vai trò của dịch vụ tên

Trong đó:

- Server đăng ký tên cho đối tượng có thể được gọi từ xa của mình với Dịch vụ danh bạ (Registry Server).
- Client tìm đối tượng ở xa thông qua tên đã được đăng ký trên Registry Server (looks up) và tiếp đó gọi các phương thức ở xa.

Hình 5.2. minh họa cũng cho thấy cách thức mà hệ thống RMI sử dụng một WebServer sẵn có để truyền tải mã bytecodes của các lớp qua lại giữa Client và Server

Tiến trình vận hành của một ứng dụng Client/Server theo kiểu RMI diễn ra như sau:

- Bước 1: Server tạo ra các đối tượng cho phép gọi từ xa cùng với các Stub và Skeleton của chúng.
- Bước 2: Server sử dụng lớp Naming để đăng ký tên cho một đối tượng từ xa (1).
- Bước 3: Naming đăng ký Stub của đối tượng từ xa với Registry Server (2).
- Bước 4: Registry Server sẵn sàng cung cấp tham thảo đến đối tượng từ xa khi có yêu cầu (3).
- Client yêu cầu Naming định vị đối tượng xa qua tên đã được đăng ký (phương thức lookup) với dịch vụ tên (4).
- Naming tải Stub của đối tượng xa từ dịch vụ tên mà đối tượng xa đã đăng ký về Client (5).
- Cài đặt đối tượng Stub và trả về tham khảo đối tượng xa cho Client (6).
- Client thực thi một lời gọi phương thức từ xa thông qua đối tượng Stub (7).

3. Các lớp hỗ trợ lập trình với RMI

Java hỗ trợ các lớp cần thiết để cài đặt các ứng dụng Client-Server theo kiểu RMI trong các gói: java.rmi. Trong số đó các lớp thường được dùng là:

- java.rmi.Naming:

- java.rmi.RMISecurityManager
- java.rmi.RemoteException;
- java.rmi.server.RemoteObject
- java.rmi.server.RemoteServer
- java.rmi. server.UnicastRemoteObject

II. XÂY DỰNG CHƯƠNG TRÌNH PHÂN TÁN RMI

Xây dựng một ứng dụng phân tán bằng cơ chế RMI gồm các bước sau:

1. Thiết kế và cài đặt các thành phần của ứng dụng.
2. Biên dịch các chương trình nguồn và tạo ra Stub và Skeleton.
3. Tạo các lớp có thể truy xuất từ mạng cần thiết.
4. Khởi tạo ứng dụng

1. Kỹ thuật lập trình RMI

Đầu tiên chúng ta phải xác định lớp nào là lớp cục bộ, lớp nào là lớp được gọi từ xa. Nó bao gồm các bước sau:

- *Định nghĩa các giao diện cho các phương thức ở xa (remote interfaces)*: Một remote interface mô tả các phương thức mà nó có thể được kích hoạt từ xa bởi các Client. Đi cùng với việc định nghĩa Remote Interface là việc xác định các lớp cục bộ làm tham số hay giá trị trả về của các phương thức được gọi từ xa.
- *Cài đặt các đối tượng từ xa (remote objects)*: Các Remote Object phải cài đặt cho một hoặc nhiều Remote Interfaces đã được định nghĩa. Các lớp của Remote Object class cài đặt cho các phương thức được gọi từ xa đã được khai báo trong Remote Interface và có thể định nghĩa và cài đặt cho cả các phương thức được sử dụng cục bộ. Nếu có các lớp làm đối số hay giá trị trả về cho các phương thức được gọi từ xa thì ta cũng định nghĩa và cài đặt chúng.
- *Cài đặt các chương trình Client*: Các chương trình Client có sử dụng các Remote Object có thể được cài đặt ở bất kỳ thời điểm nào sau khi các Remote Interface đã được định nghĩa.

Để nắm được kỹ thuật lập trình RMI cụ thể chúng ta xây dựng chương trình đơn giản sử dụng RMI như sau: Viết chương trình ứng dụng phân tán RMI theo mô hình client server. Chương trình có cấu trúc sau:

- Chương trình server: có đối tượng có phương thức cho phép gọi từ xa int add(int x,int y) để tính tổng của 2 số.
- Chương trình client: cho phép gọi phương thức thừ xa add() qua mạng để tính tổng của 2 số nguyên và hiển thị kết quả.

Quá trình xây dựng chương trình này có thể thực hiện qua các bước sau:

Bước 1: Khai báo giao diện để khai báo các phương thức cho phép gọi từ xa. Vì trong kỹ thuật RMI, bất kể phương thức nào cho phép gọi từ xa qua mạng đều phải được khai báo trong giao diện kế thừa từ giao diện Remote thuộc lớp java.rmi. Và phương thức đó phải đảm bảo 2 yêu cầu sau:

- Phải có thuộc tính public
- Phải ném trả về ngoại lệ RemoteException

Giả sử giao diện có tên TT, chúng ta có thể khai báo nó như sau:

```
//TT.java
import  java.rmi.*;
public interface   TT   extends     Remote
{
public int   add(int x,int  y) throws    RemoteException;
}
```

Bước 2: Khai báo lớp thực thi giao diện TT để cài dặt phương thức add(). Giả sử lớp có tên là TTImpl:

```
//TTImpl.java
import java.rmi.*;
class  TTImpl  implements   TT
{
public int add(int x,int y) throws   RemoteException
{
return  (x+y);
}
}
```

Bước 3: Xây dựng chương trình server. Chương trình server phải thực hiện 3 vấn đề cốt lõi sau đây:

- Tạo đối tượng có phương thức cho phép gọi từ xa và trả về tham chiếu đến giao diện của chúng. Ví dụ:

  TT c=new TTImpl();

- Đăng ký đối tượng có phương thức cho phép gọi từ xa với máy ảo java, thông qua trình đăng ký của JDK hoặc do người lập trình tự định nghĩa, bằng cách sử dụng phương thức *exportObject()* của lớp UnicastRemoteObject thuộc gói java.rmi.server. Phương thưc *exportObject()* có thể khai báo như sau:

  UnicastRemoteObject.exportObject(Obj);

- Gán cho đối tượng có phương thức cho phép gọi từ xa một tên dưới dạng chuỗi URL để thông qua chuỗi tên đó, các đối tượng client có thể truy xuất tìm thấy đối tượng trên

mạng. Để thực hiện việc đó, lớp java.rmi.Naming cung cấp phương thức *bind()* hoặc *rebind()*. Phương thức *bind()* có dạng sau:

Naming.bind("rmi//<host>[:port]/ObjName", Obj);

Chương trình server được viết như sau:

```
//TTServer.java
import java.rmi.*;
import java.rmi.serrver.*;
class TTServer{
public static void main(String[] args)
{
try{
//Tao doi tuong
TT   c=new   TTImpl();
//dang ky voi mays ao java
UnicastRemoteObject.exportObject(c);
//Gan chuoi URL
Naming.bind("rmi://localhost/Obj", c);
System.out.println("Server RMI da san sang....");
}
catch(Exception   e)
{
System.out.println(e);
}
}}
```

Bước 4: Xây dựng chương trình client, giả sử chương trình là lớp TTClient.java.. Chương trình client phải có nhiệm vụ sau:

- Truy xuất tìm đối tượng có phương thức cho phép gọi từ xa thông qua chuỗi tên URL đa được chương trình server gán cho đối tượng. Bằng cách client sử dụng phương thức *lookup()* của lớp Naming hỏi bộ đăng ký thông qua số cổng cụ thể đã được định nghĩa trong chuỗi URL. Nếu tìm thấy, server sẽ trả về tham chiếu đến đối tượng từ xa có kiểu giao diện của đối tượng.
- Gọi thi hành phương thức từ xa thông qua biến tham chiếu tới đối tượng từ xa.

Chương trình client:

```
//TTClient.java
import java.rmi.*;
class TTClient{
public static void main(String[] args)
```

```
{
try{
TT   x=(TT)Naming.lookup("rmi://localhost/Obj");
int a=10, b=20;
System.out.println("Tong cua a="+a+" voi b="+b+" la s="+x.add(a,b));
}
catch(Exception e)
{
System.out.println(e);
}
}}
```

2. Biên dịch chương trình

Giai đoạn này gồm 2 bước:

Bước thứ nhất: Biên dịch các tệp chương trình thành dạng bytecode dùng trình javac.exe. Trong chương trình trên có 4 tệp:

javac.exe

TT.java ----------->TT.class (1)

TTImpl.java ----------->TTImpl.class (2)

TTServer.java ----------->TTServer.class (3)

TTClient.java ----------->TTClient.class (4)

Bước thứ 2: Phát sinh các tệp đối tượng trung gian _stub và _skel bằng cách sử dụng trình dịch rmic.exe của JDK để dịch tệp đối tượng có phương thức cho phép gọi từ xa TTImpl:

rmic TTImpl [Enter]

Sau khi dich, 2 tệp mới được tạo ra:

TTImpl_Stub.class (5)

TTImpl_Skel.class (6)

3. Thực thi chương trình ứng dụng

Bước 1: Phân bố các tệp chương trình phù hợp từ (1) đến (6) về máy client và server. Cụ thể:

- Phía client: (1), (4), (5)
- Phía server: (1), (2), (3), (5), (6)

Bước 2: Chạy chương trình

Thực hiện mở 3 của sổ lênh:

- Cửa sổ thứ nhất: Chạy trình đăng ký rmiregistry.exe với cú pháp sau:

rmiregistry [porrt] [Enter]

- Của sổ thứ 2: Chạy chương trình server:

  java TTServer [Enter]

- Của sổ thứ 3: Chạy chương trình client:

  java TTClient [Enter]

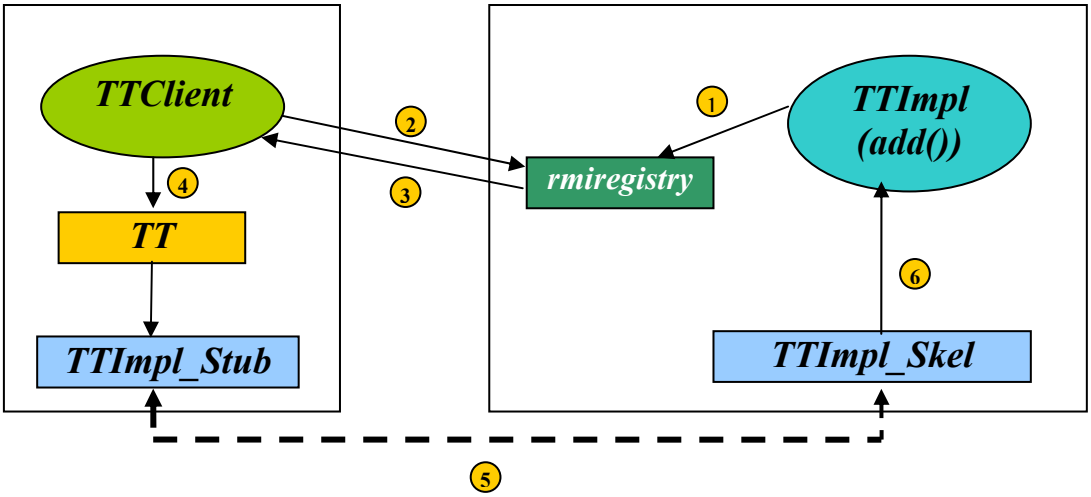
Sau khi thực hiện xong, sửa lại chương trình client phần địa chỉ host trong chuỗi URL, dịch lại và có thể chạy thử qua môi trường mạng. Khi đó của sổ 1, 2 chạy phia bên server, cửa sổ 3 chạy phía client.

III. CƠ CHẾ TRUYỀN THÔNG RMI

Cơ chế truyền thông RMI có thể giải thích theo hình 5.3 và nó thực hiện theo các bước sau:

Step 1: Đầu tiên đối tượng cài đặt các phương thức và gọi hàm *Naming.bind()* để đăng kỹ với bộ quản lý rmiregistry trên server thông qua quá trình 1.

Step 2: Đối tượng trên client gọi hàm Naming.lookup() để truy tìm đối tượng từ xa bằng cách hỏi bộ đăng ký thông qua chuỗi URL bằng quá trình 2.



Hình 5.3. Cơ chế RMI gọi phương thức của đối tượng từ xa

Step 3: Bộ đăng ký rmiregistry tìm đối tượng, nếu thấy nó trả về tham chiếu đến đối tượng từ xa cho client bằng quá trình 3 thông qua lớp giao diện(interface) mà đối tượng từ xa cung cấp.

Step 4: Dựa vào giao diện(TT) đối tượng TTClient sẽ gọi phương thức từ xa của đối tượng trên server(TTImpl) thông qua tham chiếu nhận được ở bước 3 bằng quá trình 4.

Step 5: Khi một phương thức được gọi, lời gọi sẽ được chuyển tới đối tượng trung gian _Stub và được đóng gói chuyển qua mạng theo giao thức JRMP tời đối tượng _Skel phía server.

Step 6: Đối tượng Skel phía server sẽ khôi phục lại lời gọi và gọi thi hành phương thức từ xa bằng quá trình 6.

Step 7: Sau khi phương thức tư xa thi hành xong, kết quả sẽ được đối tượng _Skel trả về cho đối tượng client bằng một quá trình truyền thông ngược với quá trình trên.

IV. VẤN ĐỀ TRUYỀN THAM SỐ CHO PHƯƠNG THỨC GỌI TỪ XA

1. Giới thiệu truyền tham số theo tham trị và tham chiếu cho phương thức từ xa

Trong kỹ thuật RMI, việc truyền tham số trong các lời gọi các phương thức từ xa qua mạng cũng có 2 cách: truyền tham trị và truyền tham chiếu. Với các tham số truyền có kiểu dữ liệu cơ bản thì đều là truyền tham trị. Còn đối với các đối tượng thì có 2 kiểu truyền: truyền đối tượng theo kiểu tham trị và truyền đối tượng theo kiểu tham chiếu. Trong cách truyền đối tượng theo kiểu tham trị thì bản thân đối tượng sẽ được truyền qua mạng. Cách truyền này có hạn chế là:

- Ảnh hưởng đến tốc độ của mạng nhất là khi truyền đối tượng có kích thước lớn.
- Chỉ truy xuất gọi phương thức từ xa theo một chiều từ client tới server.

Trong cách truyền đối tượng theo kiểu tham chiếu thì chỉ có tham chiếu đến đối tượng được truyền qua mạng nên khắc phục được hạn chế của truyền đối tượng theo kiểu tham trị. Và nó cho phép truy xuất gọi phương thức từ xa theo cả 2 chiều từ client đến server và ngược lại.

Để chỉ thị một đối tượng truyền cho phương thức từ xa qua mạng là truyền tham trị hay truyền tham chiếu thì trong Java sử dụng cách cài đặt như sau: Tất cả các đối tượng có kiểu lớp thực thi giao diện Serializable thì khi truyền cho phương thức đều được ấn định là truyền tham trị. Còn các đối tượng mà có kiểu lớp thực thi giao diện Remote thì khi truyền cho phương thức từ xa sẽ là truyền tham chiếu.

Sau đây chúng ta sẽ khảo sát kỹ 2 cách truyền đối tượng theo kiểu tham trị và kiểu tham chiếu.

2. Truyền đối tượng theo kiểu tham trị

Để nắm được kỹ thuật này, chúng ta xét ví dụ sau: Viết chương trình RMI có cấu trúc sau:

- Phía client cho phép tạo đối tượng BOX có các tham số w, h, d tương ứng là chiều rộng, chiều cao và chiều sâu của hình hộp chữ nhật. Sau đó gọi phương thức từ xa và truyền đối tượng BOX cho phương thức theo kiểu tham trị, nhận kết quả trả về và hiển thị.
- Phía server có đối tượng có phương thức cho phép gọi từ xa với tham số truyền là đối tượng BOX, thực hiện thay đổi w, h, d của đối tượng và trả đối tượng về cho client.

Quá trình xây dựng chương trình thực hiện các bước sau:

Bước 1: Xây dựng lớp BOX thực thi giao diện Serializable thuộc gói java.io để đối tượng có thể truyền theo kiểu tham trị.

```
//BOX.java
import java.io.*:
class  BOX  implements  Serializable
{
int   w,h,d;
```

```
BOX(){
w=10; h=20; d=15;
}}
```

Các bước 2 trở đi tương tự như kỹ thuật xây dựng chương trình RMI đã khảo sát ở phần trên.

Bước 2: Xây dựng giao diện để khai báo phương thức changeOBject() cho phép gọi từ xa

```
//BB.java
import java.rmi.*;
interface BB  extends   Remote
{
public  BOX   changeObject(BOX     obj) throws   RemoteException;
}
```

Bước 3: Khai báo lớp thực thi giao diện BB

```
//BBImpl.java
import java.rmi.*;
class  BBImpl   implements   BB
{
public  BOX  changeObject(BOX   obj) throws  RemoteException
{
obj.w+=10; obj.h+=5; obj.d+=15;
return obj;
}}
```

Bước 4: Xây dựng chương trình server

```
//BBServer.java
import java.rmi.*;
import java.rmi.server.*;
class BBServer{
public static boid main(String[] args)
{
try{
BB  c=new   BBImpl();
UnicastRemoteObject.exportObject(c);
Naming.bind("rmi://localhost/cObj",c);
}
catch(Exception  e)
{
System.out.println(e);
```

```
}}}
```

Bước 5: Xây dựng chương tình client

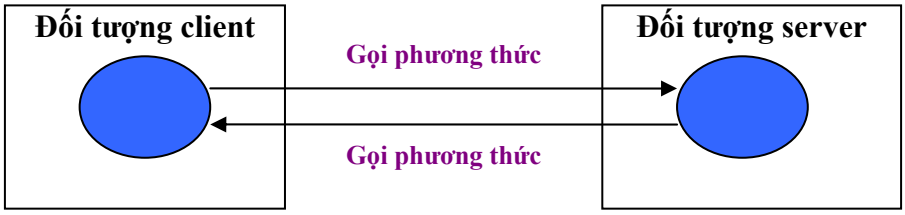
```
//BBClient.java
import  java.rmi.*;
class  BBClient{
public static void main(String[]  args)
{
try{
BB  c=(BB)Naming.lookup("rmi://localhost/cObj");
BOX  box=new  BOX();
System.out.println("w="+box.w+",h="+box.h+",d="+box.d);
box=c.changeObject(box);
System.out.println("w="+box.w+",h="+box.h+",d="+box.d);
}
catch(Exception e)
{
System.out.println(e);
}
}}
```

Sau khi tạo chương trình xong, thực hiện dịch và chạy chương trình theo kỹ thuật đã được trình bày ở phần trên. Để rõ hơn, các bạn có thể hiển thị kích cỡ của đối tượng BOX phía client, hiện thị kích cỡ đối tượng đó sau khi đã gọi phương thức từ xa phía server để so sánh.

3. Truyền đối tượng theo kiểu tham chiếu

Trong ký thuật này, đối tượng phía bên client và server đều có thể gọi phương thức từ xa của nhau và khác với kỹ thuật truền tham số trên là chỉ truyền tham chiếu đến đối tượng thay vì truyền bản thân đối tượng.

Nhưng có một vấn đề cần lưu ý trong kỹ thuật này là: Đối tượng phía client gọi phương thức từ xa của đối tượng phía server thì vẫn xẩy ra như đã trình bày. Nhưng khi đối tượng server gọi phương thức từ xa của client thí cơ chế có khác một chút. Cơ chế gọi ngược từ xa của đối tượng trên server đến đối tượng trên client thông quan tham chiếu gọi là cơ chế callback.



Hình 5.4. Đối tượng giữa client và server gọi phương thức của nhau

Để thể hiện kỹ thuật truyền đối tượng theo kiểu tham chiếu, chúng ta sẽ xét ví dụ sau[2]: Chúng ta sẽ tạo ra 2 đối tượng là AtClient chạy trên client và AtServer chạy trên server. Đầu tiên như kỹ thuật rmi bình thường, client sẽ hỏi trình rmiregistry để timg tham chiếu đến đối tượng AtServer, sau đó trình client sẽ tạo đối tượng AtClient phía client và gọi phương thức của AtServer để đăng ký đối tượng AtClient với server. Sau khi thực hiện những thao tác này, đối tượng AtClient và AtServer có thể tự do điều khiênt và gọi phương thức từ xa của nhau. Quá trình thực hiện này có thể thể hiện thông qua các bước sau:

Bước 1: Xây dựng giao diện phias client là AtClient.

```
//AtClient.java
import java.rmi.*;
public interface  AtClient  extends  Remote
{
public void callClientMethod(String msg) throws  RemoteException;
}
```

Bước 2: Xây dựng giao diện phía server là AtServer

```
//AtServer.java
import java.rmi.*;
public interface  AtServer  extends  Remote
{
public void registerClient(AtClient  c)throws  RemoteException;
public void callServerMethod(String  msg) throws RemoteException;
}
```

Bước 3: Cài đặt lớp thực thi cho các giao diện AtClient và AtServer

```
//AtClientImpl.java
import java.rmi.*;
class AtClientImpl  implements   AtClient
{
public void  callClientMethod(String msg) throws RemoteException
{
System.out.println(msg);
}}
//AtServerImpl.java
import java.rmi.*;
class AtServerImpl  implements   AtServer
{
AtClient client;
public void registerClient(AtClient c) throws RemoteException{
```

```
client=c;
}
public void  callServerMethod(String msg) throws RemoteException
{
System.out.println(msg);
for(int i=0;i<10;i++){
String msg="Server response  "+Math.random()*1000;
client.callClientMethod(msg);
}}}
```

Bước 4: Xây dựng chương trình server

```
//rServer.java
import java.rmi.*;
import java.rmi.server.*;
class  rServer{
public static void main(String[] args) throws Exception
{
AtServer  server=new  AtServerImpl();
UnicastRemoteObject.exportObject(server);
Naming.bind("rmi://localhost/serverObject",server);
System.out.println("Waiting for client request...");
}}
```

Bước 5: Xây dựng chương trình client

```
//rClient.java
import java.rmi.*;
import java.rmi.server.*;
class  rClient{
public static void main(String[] args) throws Exception
{
AtClient  cl=new  AtClientImpl();
UnicastRemoteObject.exportObject(cl);
Naming.bind("rmi://localhost/clObject", cl);
AtServer svr=(AtServer)Naming.lookup("rmi://localhost/serverObject");
svr.registerClient(cl);
svr.callServerMethod("Client contact server");
}}
```

Bước 6: Giả sử các tệp chương trình trên đều đặt trong thư mục D:\rmi, quá trình dịch và chạy chương trình có thể thực hiện bằng các câu lệnh sau:

- Dịch:

  *D:\rmi\>javac *.java*

` *D:\rmi\>rmic AtServerImpl*

  D:\rmi\>rmic AtClientImpl

- Mở của sổ lệnh chạy trình đăng ký

  D:\rmi\>rmiregistry

- Mở cửa sổ lệnh chạy trình rServer

  D:\rmi\>java rServer

- Mở cửa sổ lệnh chạy trình rClient

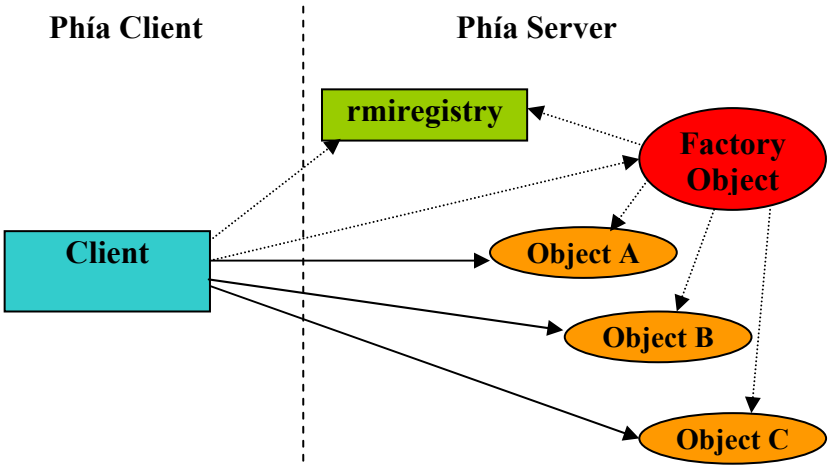
  D:\rmi\>java rClient

V. KỸ THUẬT SỬ DỤNG MỘT ĐỐI TƯỢNG SẢN SINH NHIỀU ĐỐI TƯỢNG

1. Giới thiệu

Chúng ta đã biết, một đối tượng có phương thức cho phép gọi từ xa trưwcs khi sử dụng phải đăng ký đối tượng với máy ảo java thông qua trình rmiregistry, sau đó phải gán một chuỗi URL phía server và phía client phải có lệnh tìm kiếm thông qua URL đó. Đối với chương trình đơn giản, ít đối tượng thì không vẫn đề gì. Nhưng khi phía Server có hàng trăm, hàng nghìn và hơn nữa các đối tượng có phương thức cho phép gọi từ xa thi vẫn đề trở nên nghiêm trọng. Khi đó trình rmiregistry phải quản lý qua nhiều đối tượng, người lập trình chương trình client phải nhớ nhiều đối tượng, số câu lệnh đăng kỹ đối tượng, gán chuỗi URL phía server(bind()) và câu lệnh tìm kiếm phía client(lookup()) quá nhiều, tương tác qua mạng quá nhiều, từ đó chương trình trở nên phức tạp. Để giải quyết vấn đề này, rmi có một kỹ thuật cực kỳ hữu ích. Đó là, thay vì tạo truy xuất, nhớ nhiều đối tượng thì bầy giờ người lập trình chỉ cần nhớ một đối tượng, chỉ cần đăng ký với máy ảo java một đối tượng và chỉ cần truy tìm một đối tượng. Còn tất cả các đối tượng còn lại sẽ do đối tượng đại diện này tạo ra, đang ký và cho phép phía client chỉ cần gọi các phương thức từ xa của đối tượng đại diện để trả về tham chiếu đến các đối tượng do nó sản sinh ra. Đối tượng đó gọi là đối tượng sản sinh nhiều đối tượng(Factory Object). Hình 5.5. cho chung ta thấy cơ chế sử dụng một đối tượng(Factory Object) để sản sinh nhiều đối tượng khác(A, B, C). Cơ chế sử dụng đối tượng Factory để sản sinh nhiều đối tượng khác được thực hiện như sau:

- Đầu tiên đối tượng Factory Object đăng kỹ với bộ đăng ký rmiregistry
- Trình client muốn gọi đối tượng A, B, C; trước hết trình client phải liên hệ với rmiregistry để lấy về tham chiếu đến đối tượng Factory Object.
- Sau khi có tham chiếu đến đối tượng Factory Object, trình client thực hiện Factory Object để yêu cầu tạo ra các đối tượng A, B, C, đăng ký các đối tượng A, B, C với máy ảo java và trả về tham chiếu đến A, B, C cho client.
- Dựa vào tham chiếu nhận được, client thực hiện lời gọi phương thức từ xa của các đối tượng A, B, C.



Hình 5.5. Mô hình hoạt động của đối tượng Factory

2. Kỹ thuật cài đặt ứng dụng Factory

Kỹ thuật cài đặt đối tượng sản sinh ra nhiều đối tượng được thể hiện thông qua ví dụ sau: Giả sử mô hình bài toán như hình 5.5. Trong đó đối tượng Factory Object có các phương thức từ xa cho phép trả về tham chiếu đến đối tượng A, B, C là getRef_X() sau khi đối tượng này đã tạo A, B, C và đăng ký với rmiregistry. Sau đó client sẽ gọi các phương thức từ xa của các đối tượng A, B, C laf get_X(). Các phương thức này sẽ trả về cho client chuỗi "Day la doi tượng X". Với X là A, B hoặc C. Giả sử giao diện của đối tượng Factory Object là FF và lớp thực thi là FFImpl.

Bước 1: Xây dựng giao diện của A, B, C tương ứng là AA, BB, CC

```
//AA.java
import java.rmi.*;
public interface  AA  extends   Remote
{
public String   get_A()  throws  RemoteException;
}
//BB.java
import java.rmi.*;
public interface  BB  extends   Remote
{
public String   get_B()  throws  RemoteException;
}
//CC.java
import java.rmi.*;
public interface  CC  extends   Remote
```

```
{
public String   get_C()  throws  RemoteException;
}
```

Bước 2: Khai báo các lớp thực thi các giao diện AA, BB, CC

```
//AAImp.java
import java.rmi.*;
class  AAImpl  implements   AA{
public String   get_A()  throws  RemoteException{
return   "Day la doi tuong A.";
}}
//BBImpl.java
import java.rmi.*;
class  BBImpl  implements   BB{
public String   get_B()  throws  RemoteException{
return   "Day la doi tuong B.";
}}
//CCImpl.java
import java.rmi.*;
class  CCImpl  implements CC{
public String   get_C()  throws  RemoteException{
return   "Day la doi tuong C.";
}}
```

Bước 3: Khai báo giao diện FF, trong giao diện này khai báo các phương thức getRef_X() để trả về các tham chiếu đến đối tượng A, B, C

```
//FF.java
import   java.rmi.*;
public interface   FF   extends    Remote{
public AA  getRef_A() throws   RemoteException;
public BB  getRef_B() throws   RemoteException;
public CC  getRef_C() throws   RemoteException;
}
```

Bước 5: Khai báo lớp FFImpl thực thi giao diện FF. Trong lớp này phải thực hiện các công việc sau:

- Tạo các đối tượng A, B, C tương ứng là AAImpl, BBImpl, CCImpl
- Đăng ký các đối tượng này với trình đăng ký rmiregistry
- Cài đặt các phương thức trả về tham chiếu đế các đối tượng đã được tạo ra

```
//FFImpl.java
```

```
import  java.rmi.*;
import  java.rmi.serrver.*;
class  FFImpl   implements   FF{
//Tao cac doi tuong
AA   a=new    AAImpl();
BB   b=new    BBImpl();
CC    c=new   CCImpl();
//Khai bao cau tu FFImpl, trong do thuc hien dang ky cac doi tuong
FFImpl()
{
try{
UnicastRemoteObject.exportObject(a);
UnicastRemoteObject.exportObject(b);
UnicastRemoteObject.exportObject(c);
}catch(Exception  e)
{}
}
//Cai dat cac phuong thu
public AA  getRef_A() throws  RemoteException{
return    a;
}
public BB  getRef_B() throws  RemoteException{
return    b;
}
public CC getRef_C() throws  RemoteException{
return    c;
}
}
```

Bước 6: Xây dựng chương trình phía server. Chương trình này phải thực hiện các nhiệm vụ sau(như kỹ thuật có bản):

- Tạo đối tượng FFImpl
- Đăng kỹ đối tượng FFImpl với máy ảo java
- Gán cho đối tượng một chuỗi URL để truy xuất đối tượng trên mạng

```
//FFServer.java
import  java.rmi.*;
import  java.rmi.server.*;
```

```
class  FFServer{
public static void main(String[] args) throws  Exception
{
FF    f=new    FFImpl();
UnicastRemoteObject.exportObject(f);
Naming.bind("rmi://localhost/ffObj",f);
System.out.println("Server da san sang...");
}}
```

Bước 7: Xây dựng chương trình client. Chương trình này có nhiệm vụ sau:

- Tìm đối tượng FFImpl và nhận tham chiếu đến đối tượng FFImpl
- Gọi các phương thức của FFImpl để lấy tham chiếu đến các đối tượng A, B, C
- Thông qua tham chiếu đến A, B, C gọi từ xa các phương thức của các đối tượng này.

```
//FFClient.java
import   java.rmi.*;
class   FFClient{
public static void main(String[] args ) throws Exception
{
//Lay tham chieu doi tuong Factorry
FF   f=(FF)Naming.lookup("rmi://localhost/ffObj");
//Goi phuong thuc cua Factory tra ve tham chieu toi A, B, C
AA  a=f.getRef_A();
BB  b=f.getRef_B();
CC  c=f.getRef_C();
//Goi cac phuong thuc cuar A, B, C thong qua tham chieu
System.out.println(a.get_A());
System.out.println(b.get_B());
System.out.println(c.get_C());
}}
```

Bước 8: Dịch và chạy chương trình

//Giả sử tất cả các tệp nguồn nằm trong thư mục d:\RMI\Factory

*D:\RMI\Factory>javac *.java*

//Phat sinh các tệp _Stub, _Skel của FFImp, AAImpl, BBImpl, CCImpl

D:\RMI\Factory>rmic AAImpl

D:\RMI\Factory>rmic BBImpl

D:\RMI\Factory>rmic CCImpl

D:\RMI\Factory>rmic FFImpl

//Chạy chương trình

//b1: Mở cửa sổ lệnh chạy trình đăng ký

D:\RMI\Factory>rmiregistry

//b2: Mở cửa sổ lệnh chạy chương trình server

D:\RMI\Factory>java FFServer

//b3: Mở cửa sổ lệnh chạy chương trình client

D:\RMI\Factory>java FFClient

Lưu ý: Nếu chạy chương trình server và client trên 2 máy tính nối mạng thì phải phân bố các tệp server và các tệp phía client tương tự như kỹ thuật có bản nhưng nhiều đối tượng. Và phải đổi địa chỉ trong chuỗi URL phía chương trình client là địa chỉ của server(trong lookup()).

VI. KẾT LUẬN

Qua các mục của chương này, chúng ta đã làm sáng tỏ kỹ thuật lập trình, cơ chế truyền thông và bản chất của lập trình phân tán đối tượng của RMI. Thông qua đó, sinh viên có thể hiểu đực các kỹ thuật lập trình khác như RPC, DCOM, CORBA, EJB, WebService... với các kỹ thuật lập trình OOP, SOP và kiến trúc nhiều tầng. Ngoài các vấn đề nêu trong chương, còn một số kỹ thuật khác của RMI không kém phần quan trọng mà sẽ được đề cập đến trong bài giảng và thông qua bài tập của sinh viên như: Vấn đề định nghĩa bộ đăng ký, vấn đề tuần tự hoá đối tượng, Kỹ thuật gọi đối tượng từ xa bằng phương thức động, kỹ thuật kích hoạt đối tượng từ xa tụ động, chính sách bảo mật từ phía client.v.v..