

# CHƯƠNG 5

## RPC và RMI

### Mục đích

Chương này nhằm giới thiệu cách thức xây dựng các ứng dụng phân tán bằng các cơ chế gọi thủ tục từ xa (RPC - Remote Procedure Call và RMI - Remote Method Invocation)

### Yêu cầu

Sau khi hoàn tất chương này, bạn có thể:

- Định nghĩa được ứng dụng phân tán là gì.
- Trình bày được kiến trúc của một ứng dụng phân tán xây dựng theo cơ chế gọi thủ tục từ xa (RPC).
- Trình bày được kiến trúc của một ứng dụng phân tán (hay còn gọi Ứng dụng đối tượng phân tán ) xây dựng theo cơ chế RMI của Java.
- Trình bày được các cơ chế liên quan khi xây dựng một ứng dụng theo kiểu RMI.
- Trình bày được cơ chế vận hành của một ứng dụng theo kiểu RMI.
- Giải thích được vai trò rmiregistry server.
- Liệt kê được các lớp của java hỗ trợ xây dựng các ứng dụng kiểu RMI.
- Trình bày chi tiết các bước phải qua khi xây dựng một ứng dụng theo kiểu RMI.
- Biên soạn, biên dịch và thực thi thành công chương trình minh họa Hello.
- Phân tích, thiết kế và cài đặt được các chương trình theo cơ chế RMI để giải quyết các vấn đề cụ thể.

### 1.1. Lời gọi thủ tục xa (RPC- Remote Procedure Call)

#### 1.1.1. Giới thiệu

Lời gọi thủ tục xa là một cơ chế cho phép một chương trình có thể gọi thực thi một thủ tục (hay hàm) trên một máy tính khác. Trong chương trình lúc này, tồn tại hai loại thủ tục: thủ tục cục bộ và thủ tục ở xa.

- Thủ tục cục bộ là thủ tục được định nghĩa, cài đặt và thực thi tại máy của chương trình.

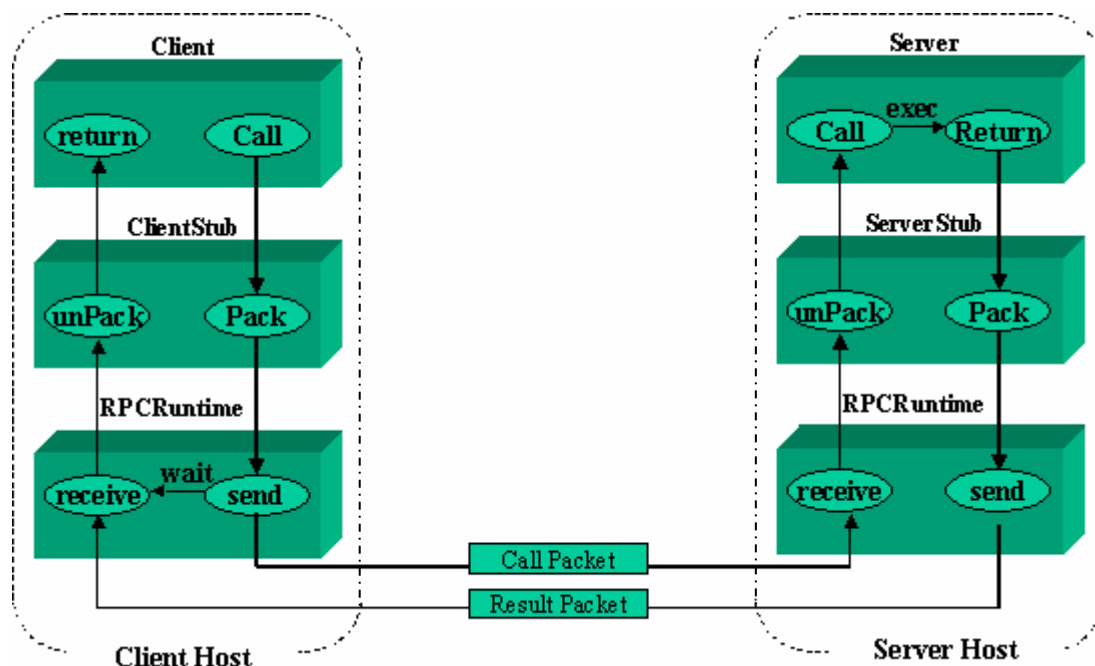
- Thủ tục ở xa là thủ tục được định nghĩa, cài đặt và thực thi trên một máy tính khác.

Cú pháp giữa lời gọi thủ tục cục bộ và ở xa thì giống nhau. Tuy nhiên, khi một thủ tục ở xa được gọi đến, một thành phần của chương trình gọi là **Stub** sẽ chuyển hướng để kích hoạt một thủ tục tương ứng nằm trên một máy tính khác với máy của chương trình gọi. Đối với người lập trình, việc gọi thủ tục xa và thủ tục cục bộ thì giống nhau về mặt cú pháp. Đây chính là cơ chế cho phép đơn giản hóa việc xây dựng các ứng dụng Client- Server. Trong hệ thống RPC, Server chính là máy tính cung cấp các thủ tục ở xa cho phép các chương trình trên các máy tính khác gọi thực hiện. Client chính là các chương trình có thể gọi các thủ tục ở xa trong quá trình tính toán của mình.

Một Client có thể gọi thủ tục ở xa của nhiều hơn một máy tính. Như vậy sự thực thi của chương trình Client lúc này không còn gói gọn trên một máy tính của Client mà nó trải rộng trên nhiều máy tính khác nhau. Đây chính là mô hình của ứng dụng phân tán (Distributed Application).

### 1.1.2. Kiến trúc của chương trình Client-Server cài đặt theo cơ chế lời gọi thủ tục xa

Một ứng dụng Client-Server theo cơ chế RPC được xây dựng gồm có sáu phần như sơ đồ dưới đây:



Hình 5.1 Kiến trúc chương trình kiểu RPC

Phần Client là một quá trình người dùng, nơi khởi tạo một lời gọi thủ tục từ xa. Mỗi lời gọi thủ tục ở xa trên phần Client sẽ kích hoạt một thủ tục cục bộ tương ứng nằm trong phần Stub của Client.

Phần ClientStub cung cấp một bộ các hàm cục bộ mà phần Client có thể gọi. Mỗi một hàm của ClientStub đại diện cho một hàm ở xa được cài đặt và thực thi trên Server.

Mỗi khi một hàm nào đó của ClientStub được gọi bởi Client, ClientStub sẽ đóng gói một thông điệp để mô tả về thủ tục ở xa tương ứng mà Client muốn thực thi cùng với các tham số nếu có. Sau đó nó sẽ nhờ hệ thống RPCRuntime cục bộ gửi thông điệp này đến phần Server Stub của Server.

Phần RPCRuntime quản lý việc truyền thông điệp thông qua mạng giữa máy Client và máy Server. Nó đảm nhận việc truyền lại, báo nhận, chọn đường gói tin và mã hóa thông tin.

RPCRuntime trên máy Client nhận thông điệp yêu cầu từ ClientStub, gửi nó cho RPCRuntime trên máy Server bằng lệnh send(). Sau đó gọi lệnh wait() để chờ kết quả trả về từ Server.

Khi nhận được thông điệp từ RPCRuntime của Client gửi sang, RPCRuntime bên phía server chuyển thông điệp lên phần ServerStub.

ServerStub mở thông điệp ra xem, xác định hàm ở xa mà Client muốn thực hiện cùng với các tham số của nó. ServerStub gọi một thủ tục tương ứng nằm trên phần Server.

Khi nhận được yêu cầu của ServerStub, Server cho thực thi thủ tục được yêu cầu và gửi kết quả thực thi được cho ServerStub.

ServerStub đóng gói kết quả thực thi trong một gói tin trả lời, chuyển cho phần RPCRuntime cục bộ để nó gửi sang RPCRuntime của Client .

RPCRuntime bên phía Client chuyển gói tin trả lời nhận được cho phần ClientStub. ClientStub mở thông điệp chứa kết quả thực thi về cho Client tại vị trí phát ra lời gọi thủ tục xa.

Trong các thành phần trên, RPCRuntime được cung cấp bởi hệ thống. ClientStub và ServerStub có thể tạo ra thủ công (phải lập trình) hay có thể tạo ra bằng các công cụ cung cấp bởi hệ thống.

Cơ chế RPC được hỗ trợ bởi hầu hết các hệ điều hành mạng cũng như các ngôn ngữ lập trình.

## 1.2. Kích hoạt phương thức xa (RMI- Remote Method Invocation )

### 1.2.1. Giới thiệu

RMI là một sự cài đặt cơ chế RPC trong ngôn ngữ lập trình hướng đối tượng Java. Hệ thống RMI cho phép một đối tượng chạy trên một máy ảo Java này có thể kích hoạt một phương thức của một đối tượng đang chạy trên một máy ảo Java khác. Đối tượng có phương thức được gọi từ xa gọi là các đối tượng ở xa (Remote Object).

Một ứng dụng RMI thường bao gồm 2 phần phân biệt: Một chương trình Server và một chương trình Client.

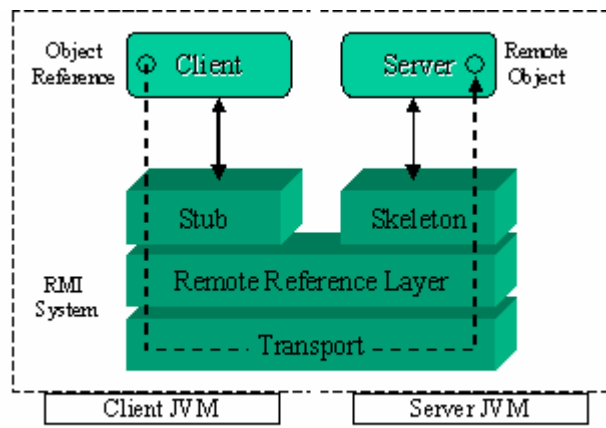
- Chương trình Server tạo một số các Remote Object, tạo các **tham chiếu** (reference) đến chúng và chờ những chương trình Client kích hoạt các phương thức của các Remote Object này.

- Chương trình Client lấy một **tham chiếu** đến một hoặc nhiều Remote Object trên Server và kích hoạt các phương thức từ xa thông qua các tham chiếu.

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.

### 1.2.2. Kiến trúc của chương trình Client-Server theo cơ chế RMI

Kiến trúc một chương trình Client-Server theo cơ chế RMI được mô tả như hình dưới đây:

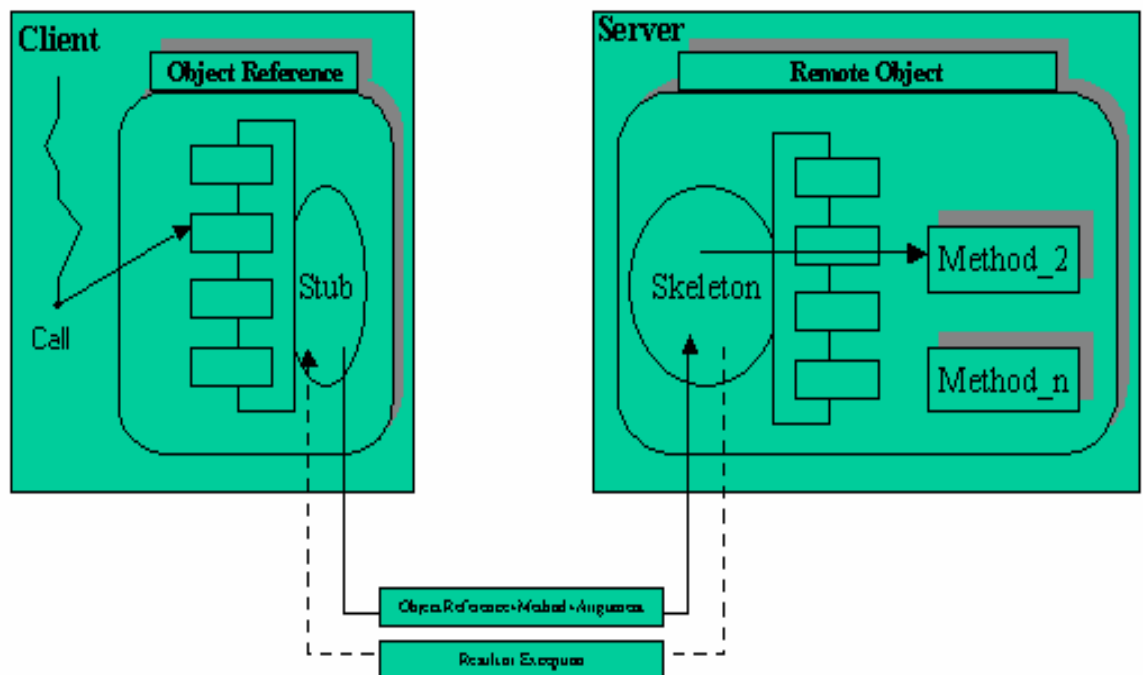


Hình 5.2 - Kiến trúc chương trình kiểu RMI

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 ở xa trên Server.
- Stub chứa các tham chiếu đến các phương thức ở xa trên Server.
- Skeleton đón nhận các tham chiếu từ Stub để kích hoạt phương thức tương ứng trên Server.
- Remote Reference Layer là hệ thống truyền thông của RMI.

Con đường kích hoạt một phương thức ở xa được mô tả như hình dưới đây:



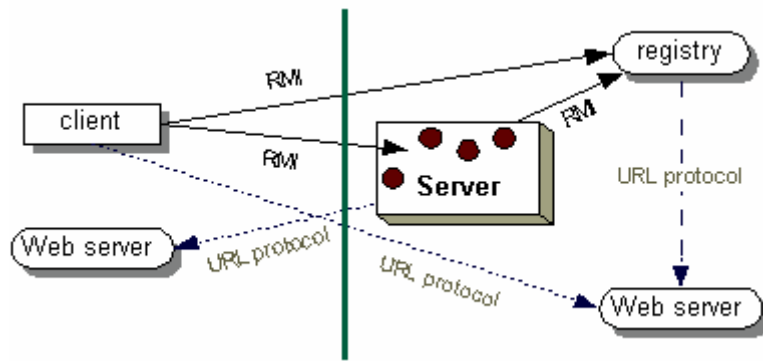
Hình 5.3 Cơ chế hoạt động của RMI

### 1.2.3. Các cơ chế liên quan trong một ứng dụng đối tượng phân tán

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

- *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 khảo đến các đối tượng cho phép gọi từ xa mà Client sau đó có thể tìm kiếm.
- *Cơ chế giao tiếp với các đối tượng ở xa (Communicate with remote objects)*: Chi tiết của cơ chế giao tiếp với các đối tượng ở xa được cài đặt bởi hệ thống RMI.
- *Tải các lớp dạng bytecodes cho các lớp mà nó được 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 ở 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 dưới đây mô tả một ứng dụng phân tán dưới RMI sử dụng dịch vụ danh bạ để lấy các tham khảo của các đối tượng ở xa.



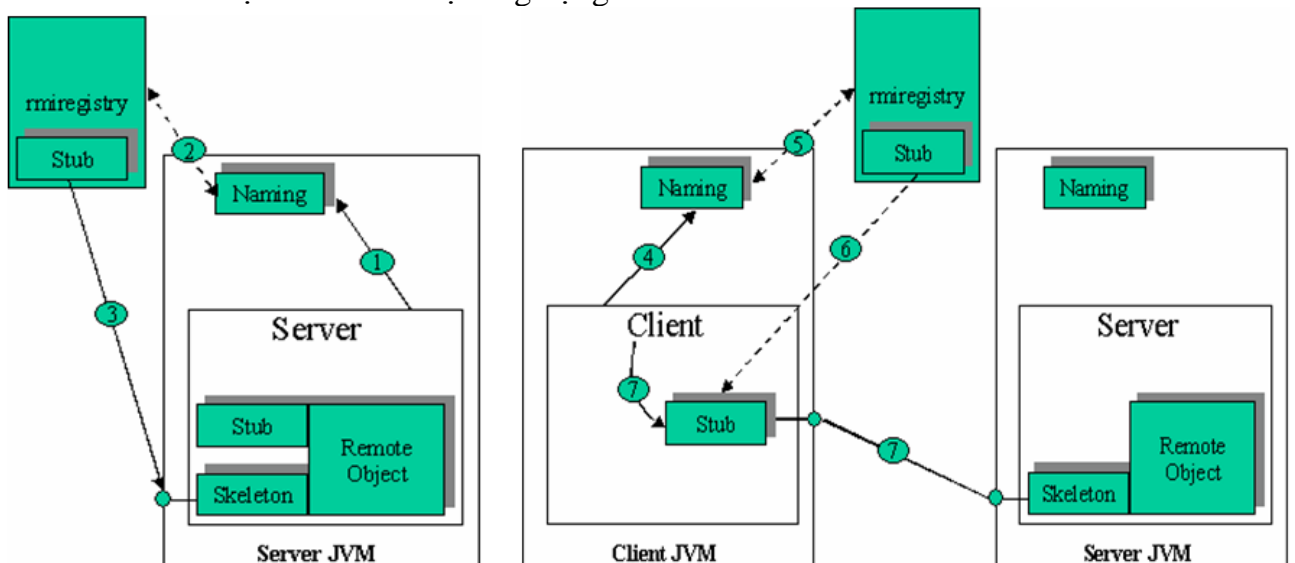
Hình 5.4 Vai trò của dịch vụ tên

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 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.

#### 1.2.4. Cơ chế vận hành của của một ứng dụng Client-Server theo kiểu RMI

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 khảo 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 xa thông qua đối tượng Stub (7).

### 1.2.5. Các lớp hỗ trợ chương trình theo kiểu Client-Server trong Java

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

## 1.3. Xây dựng một ứng dụng phân tán với RMI

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

- a. Thiết kế và cài đặt các thành phần của ứng dụng.
- b. Biên dịch các chương trình nguồn và tạo ra Stub và Skeleton.
- c. Tạo các lớp có thể truy xuất từ mạng cần thiết.
- d. Khởi tạo ứng dụng

### 1.3.1. Thiết kế và cài đặt các thành phần của ứng dụng.

Đầu tiên bạn 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.

### 1.3.2. Biên dịch các tập tin nguồn và tạo Stubs và Skeleton

Giai đoạn này gồm 2 bước: Bước thứ nhất là dùng chương trình biên dịch javac để biên dịch các tập tin nguồn như các remote interface, các lớp cài đặt cho các remote interface, lớp server, lớp client và các lớp liên quan khác. Kế tiếp ta dùng trình biên dịch rmic để tạo ra stub và skeleton cho các đối tượng từ xa từ các lớp cài đặt cho các remote interface.

### 1.3.3. Tạo các lớp có thể truy xuất từ mạng

Tạo một tập tin chứa tất cả các file có liên quan như các remote interface stub, các lớp hỗ trợ mà chúng cần thiết phải tải về Client và làm cho tập tin này có thể truy cập đến thông qua một Web server.

### 1.3.4. Thực thi ứng dụng

Thực thi ứng dụng bao gồm việc thực thi rmiregistry server, thực thi server, và thực thi client.

#### **Tóm lại các công việc phải làm là:**

- Tạo giao diện (interface) khai báo các phương thức được gọi từ xa của đối tượng.
- Tạo lớp cài đặt (implement) cho giao diện đã được khai báo.
- Viết chương trình Server.
- Viết chương trình Client.
- Dịch các tập tin nguồn theo dạng RMI để tạo ra các lớp tương ứng và stub cho client, skeleton cho server.
- Khởi động dịch vụ registry.
- Thực hiện chương trình Server.
- Thực thi chương trình Client.

### 1.3.4. Ví dụ minh họa

Trong ví dụ này chúng ta định nghĩa một phương thức String sayHello() được gọi từ xa. Mỗi khi phương thức này được kích hoạt nó sẽ trả về chuỗi "Hello World" cho Client gọi nó.

Dưới đây là các bước để xây dựng ứng dụng:

#### **Bước 01: Tạo giao diện (interface) khai báo các phương thức được gọi từ xa của đối tượng.**

*\* Cú pháp tổng quát:*

```
import java.rmi.Remote;  
import java.rmi.RemoteException;  
public interface InterfaceName extends Remote {  
    ReturnType remoteMethodOne() throws RemoteException;  
    ReturnType remoteMethodTwo() throws RemoteException;  
    ...  
}
```

*\* Định nghĩa remote interface có tên là HelloItf, có phương thức được gọi từ xa là String sayHello() như sau:*

```
import java.rmi.Remote;  
import java.rmi.RemoteException;  
public interface HelloItf extends Remote {  
    String sayHello() throws RemoteException;
```



```
}
```

Lưu chương trình này vào tập tin HelloItf.java

## **Bước 02: Tạo lớp cài đặt (implement) cho giao diện đã được khai báo:**

*\* Cú pháp tổng quát:*

```
import java.rmi. server.UnicastRemoteObject;
import java.rmi.RemoteException;
public class RemoteClass extends UnicastRemoteObject implements InterfaceName {
    public RemoteClass() throws RemoteException {
        super();
        ..... // Implement of Method
    }
    public ReturnType remoteMethodOne() throws RemoteException {
        ..... // Implement of Method
    }
    public ReturnType remoteMethodTwo() throws RemoteException {
        ..... // Definition of Method
    }
}
```

*\* Định nghĩa lớp cài đặt có tên là Hello cài đặt cho remote interface HelloItf*

```
import java.rmi. server.UnicastRemoteObject;
import java.rmi.RemoteException;
public class Hello extends UnicastRemoteObject implements HelloItf {
    public Hello() throws RemoteException {
        super();
    }
    public String sayHello() {
        return "Hello World !";
    }
}
```

Lưu chương trình này vào tập tin Hello.java

## **Bước 03: Viết chương trình Server:**

*\* Cú pháp tổng quát:*

```
import java.rmi.Naming;
import java.rmi.RemoteException;
import java.rmi.RMISecurityManager;
public class ServerName {
```

```

public static void main(String args[]) {
    if (System.getSecurityManager() == null) {    // Cài đặt cơ chế bảo mật
        System.setSecurityManager(new RMISecurityManager());
    }
    try {
        RemoteClass remoteObject = new RemoteClass();    // Tạo các đối tượng từ xa
        // Đăng ký tên cho các đối tượng từ xa
        Naming.rebind("RegistryName", remoteObject); ...
    } catch (Exception e) {
        System.out.println("Error: . . ." + e);
    }
}
}

* Tạo server có tên HelloServer chứa một đối tượng từ xa obj thuộc lớp cài đặt Hello. Đăng ký tên cho đối tượng obj là HelloObject
import java.rmi.Naming;
import java.rmi.RemoteException;
import java.rmi.RMISecurityManager;
public class HelloServer {
    public static void main(String args[]) {
        if (System.getSecurityManager() == null) {
            System.setSecurityManager(new RMISecurityManager());
        }
    }
}

```

```

        try {
            Hello obj = new Hello();
            Naming.rebind("rmi://NameServer/RegistryName", obj);
            System.out.println("HelloObject is registried");
        } catch (Exception e) {
            System.out.println("Error: " + e);
        }
    }
}

```

Lưu chương trình này vào tập tin HelloServer.java

#### **Bước 04: Viết chương trình Client:**

*\* Cú pháp tổng quát:*

```

import java.rmi.Naming;
import java.rmi.RemoteException;
public class Client {
    public static void main(String args[]) {
        String remoteObjectURL = "rmi://NameServer/RegistryName";
        InterfaceName object = null;
        try {
            object = (InterfaceName)Naming.lookup(remoteObjectURL);
            object.remoteMethodOne();
            ...
        } catch (Exception e) {
            System.out.println(" Error: "+ e);
        }
    }
}

```

*\* Tạo client có tên là HelloClient, tìm đối tượng HelloObject trên rmiregistry chẳng hạn tại địa chỉ 172.18.211.160. Gọi phương thức sayHello() và in kết quả trả về ra màn hình.*

```

import java.rmi.Naming;
import java.rmi.RemoteException;
public class HelloClient {
    public static void main(String args[]) {
        String helloURL = "rmi://localhost/HelloObject";
        HelloItf object = null;
        try {
            object = (HelloItf)Naming.lookup( helloURL);

```

```

        String message = object.sayHello();
        System.out.println(message);
    } catch (Exception e) {
        System.out.println("Client Error :" + e);
    }
}
}

```

Lưu chương trình vào tập tin HelloClient.java

**Bước 05: Dịch các tập tin nguồn theo dạng RMI để tạo ra các lớp tương ứng và stub cho client, skeleton cho server:**

*\* Cú pháp tổng quát:*

```
javac InterfaceName.java RemoteClass.java Server.java Client.java
```

(Tạo ra các lớp InterfaceName.class, RemoteClass.class, Server.class, Client.class)

```
rmic RemoteClass
```

(Tạo ra các lớp cho Skeleton và Stub: RemoteClass\_Skel.class, RemoteClass\_Stub.class)

*\* Biên dịch các lớp trong Hello:*

```
javac Hello.java HelloItf.java HelloServer.java HelloClient.java
```

```
rmic Hello
```

The screenshot shows a Windows Command Prompt window with the following text:

```

C:\> Command Prompt
D:\progs>javac Hello.java HelloItf.java HelloServer.java HelloClient.java
D:\progs>dir Hello*.class
Volume in drive D has no label.
Volume Serial Number is 5026-1F9A

Directory of D:\progs

02/08/2003  11:20a                370 Hello.class
02/08/2003  11:20a                904 HelloClient.class
02/08/2003  11:20a                215 HelloItf.class
02/08/2003  11:20a            1,049 HelloServer.class
02/04/2003  03:13a                426 HelloWorld.class
02/04/2003  03:13a            1,410 Hello_Skel.class
02/04/2003  03:13a            2,854 Hello_Stub.class
              7 File(s)              7,228 bytes
              0 Dir(s)  1,533,521,920 bytes free

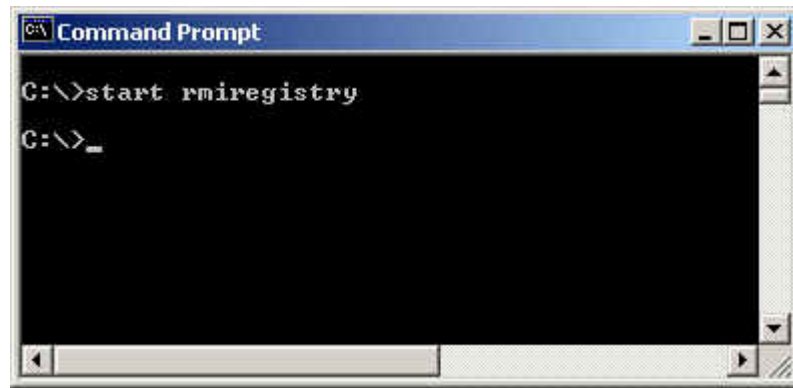
D:\progs>

```

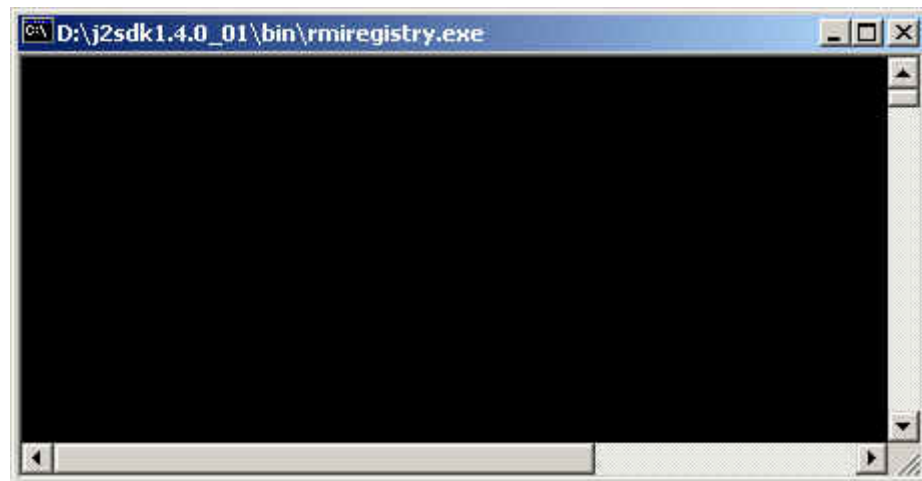
**Bước 06: Khởi động dịch vụ rmiregistry**

*\* Cú pháp tổng quát: start rmiregistry [port] Cổng mặc định là 1099.*

*\* Khởi động dịch vụ rmiregistry trên cổng mặc định như sau:*



Khi đó rmiregistry server sẽ chạy trên một cửa sổ mới, giữ nguyên cửa sổ này, không đóng nó lại.



## Bước 07: Thực hiện chương trình Server

*\* Cú pháp tổng quát:*

```
java -Djava.security.policy =UrlOfPolicyFile ServerName
```

Trong đó UrlOfPolicyFile là địa chỉ theo dạng URL của tập tin mô tả chính sách về bảo mật mã nguồn của Server (policy file). Nó qui định "ai" (chương trình, máy tính, quá trình trên) sẽ có quyền download các tập tin của nó trong đó có stub. Để đơn giản trong phần này ta cho phép tất cả mọi người đều có quyền download các tập tin của Server. Khi triển khai các ứng dụng thật sự ta phải có các chính sách bảo mật nghiêm ngặt hơn (Tham khảo tài liệu về Security của Java). File policy có dạng như sau:

```
grant {
    // Allow everything for now
    permission java.security.AllPermission;
};
```

Lưu nội dung trên vào tập tin có tên **policy.java**

*\* Thực thi HelloServer với địa tập tin ppolicy nằm ở thư mục*  
<D:\progs\policy.java>



```
Command Prompt - java -Djava.security.policy=file:///D:/progs/policy.java HelloServer
D:\progs>java -Djava.security.policy=file:///D:/progs/policy.java HelloServer
HelloObject is registried
```

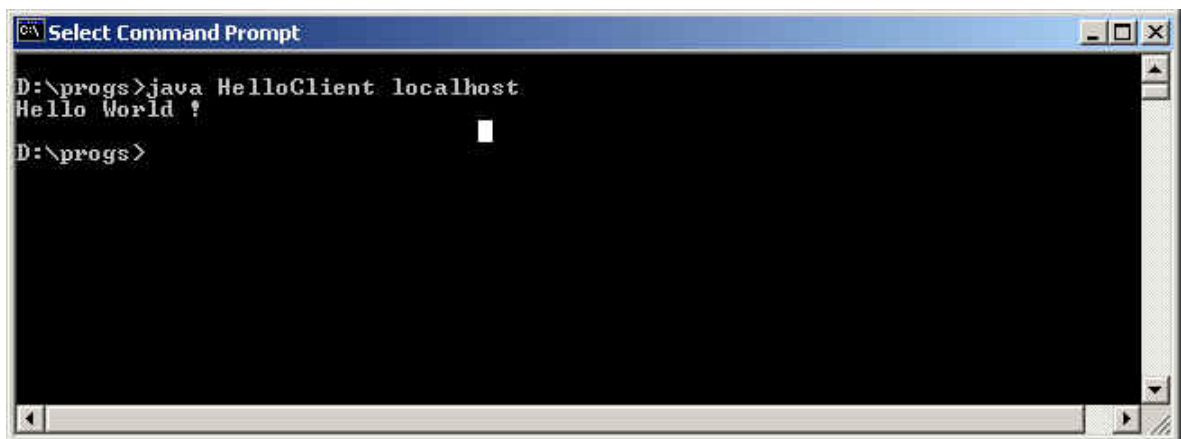
### Bước 08: Thực thi chương trình Client:

\* *Cú pháp tổng quát*

java ClientName

\* *Thực thi HelloClient với địa chỉ của rmiregistry đưa vào trong tham số*

Để thực thi được chương trình HelloClient cần có hai class nằm cùng thư mục với nó là HelloItf.class và Hello\_Stub.class.



```
Select Command Prompt
D:\progs>java HelloClient localhost
Hello World !
D:\progs>
```