CHAPTER 6

■ ■ ■

Java Remote Method Invocation

QUESTIONS AND EXERCISES

1. What is Java Remote Method Invocation?

   Answer:

   Java Remote Method Invocation (RMI) enables a Java application to invoke methods on a Java object in a remote JVM. An RMI application consists of two programs, a client and a server, that run in two different JVMs. The server program creates Java objects and makes them accessible to the remote client programs to invoke methods on those objects. The client program needs to know the location of the remote objects on the server, so it can invoke methods on them. The server program creates a remote object and registers (or binds) its reference to an RMI registry. An RMI registry is a name service that is used to bind a remote object reference to a name, so a client can get the reference of the remote object using a name-based lookup in the registry. An RMI registry runs in a separate process from the server program. It is supplied as a tool called `rmiregistry`. When you install a JDK/JRE on your machine, it is copied in the `bin` subdirectory under the JDK/JRE installation directory.

2. What is the fully qualified name of the interface that every remote interface must extend?

   Answer:

   `java.rmi.Remote`

3. What steps do you need to perform in your RMI server program after you create a remote object, so the remote object is available for a client to use?

Answer:

You need to export the remote object and register it with the RMI registry application to make the remote object available for a client to use.

4. What is RMI registry and where is it located?

Answer:

An RMI registry is a naming service. It is used by the remote server to register object using a name. The RMI server and RMI registry need to be running on the same machine. Client applications look up the remote object in the registry using the remote object name and invoke remote methods on those objects.

The RMI registry application is supplied with the JDK/JRE installation. It is copied in the `bin` subfolder of the respective installation main folder. On the Windows platform, it is the `rmiregistry.exe` executable file. You can run the RMI registry by starting the `rmiregistry` application using a command prompt. It accepts a port number on which it will run. By default, it runs on port 1099. The following command starts it at port 1099 using a command prompt on Windows:

```
C:\java9\bin> rmiregistry
```

The following command starts the RMI registry at port 8967:

```
C:\java9\bin> rmiregistry 8967
```

5. In an RMI application, can an RMI registry and RMI server be deployed to two different machines? If your answer is no, explain why.

Answer:

No. For security reasons, an RMI registry and the server must run on the same machine so that a server can register the remote references with the RMI registry. If this restriction is not imposed, a hacker may register his own harmful Java objects to your RMI registry from his machine.

6. Describe the typical sequence of steps an RMI client program needs to perform to call a method on a remote object.

Answer:

Typically, an RMI client program performs the following:

- Makes sure that it is running under a security manager.
- Locates the registry where the remote reference has been bound by the server.
- Performs the lookup in the registry using the `lookup()` method of the `Registry` interface. It passes the name of the bound remote reference to the `lookup()` method and gets back the remote reference (or stub).
- Calls methods on the remote reference (or stub). The client program treats the remote object reference as if it is a reference to a local object. Any method call made on it is sent to the server for execution.

7. An RMI application involves three layers of applications: client, RMI registry, and server. In what order must these applications be run?

Answer:

The order in which these applications should be run is as follows:

- RMI registry
- RMI server
- RMI clients

8. What is the use of the `rmic` tool? Do you need to use this tool for all RMI applications you develop?

Answer:

The `rmic` tool generates stub and skeleton classes for remote objects. You need to use the `rmic` tool only if you are using Java version prior to Java 5. You also need to perform this step if you have a client program that is running Java version prior to Java 5 and your server is running on Java 5 or later. Otherwise, stubs and skeletons are generated dynamically for you and you do not need to use this tool any more.

9. Describe the use of the `java.rmi.server.codebase` command-line option while running an RMI client and server application.

Answer:

The `java.rmi.server.codebase` option can be set in the RMI server and RMI client. The object references originating from the server or the client are annotated with the value set for this option at the server or the client, respectively. When a client works with a remote object and if the client does not find some classes locally, the client downloads those classes using the `java.rmi.server.codebase` option set at the server. If the server receives an object from the client and the server does not find some classes locally, the server downloads those classes using the `java.rmi.server.codebase` option set at the client.

10. What is the effect of using the `java.rmi.server.logCalls=true` command-line option while running an RMI server program?

Answer:

When the `java.rmi.server.logCalls` property for the server JVM is set to `true`, all incoming calls to the server and stack trace of any exceptions that are thrown during execution of an incoming call are logged to the standard error.

11. How do you log remote calls in an RMI server application to a file?

Answer:

The RMI runtime lets you log the incoming calls in a server application to a file, irrespective of the value set for the `java.rmi.server.logCalls` property for the server JVM. You can log all incoming call details to a file using the `setLog(OutputStream out)` static method of `java.rmi.server.RemoteServer` class. Typically, you set the file output stream for logging in the beginning of the server program. The following snippet of code enables the calls logging in a remote server application to a `C:\rmilogs\rmi.log` file. You can disable call logging by using `null` as the `OutputStream` in the `setLog()` method.

```
try {
    java.io.OutputStream os = new java.io.FileOutputStream("C:\\rmilogs\\rmi.log");
    java.rmi.server.RemoteServer.setLog(os);
} catch (FileNotFoundException e) {
     System.err.println("Could not enable incoming calls logging.");
     e.printStackTrace();
}
```

When a security manager is installed on the server, the running code, which enables logging to a file, must have a `java.util.logging.LoggingPermission` with permission target as "control". The following grant entry in the Java policy file will grant this permission. You will

also have to grant the "write" permission to the log file (C:\\rmilogs\\rmi.log in this example) in the Java policy file.

```
grant {
    permission java.io.FilePermission "c:\\rmilogs\\rmi.log", "write";
    permission java.util.logging.LoggingPermission "control";
};
```

12. What is the effect of using the `java.rmi.server.useCodebaseOnly=true` command-line option while running an RMI application?

Answer:

The `java.rmi.server.useCodebaseOnly` option can be set at the RMI server as well as RMI client. If this value is true, automatic loading of classes is disabled, except from the local CLASSPATH and from the `java.rmi.server.codebase` property set on the local JVM. This property is set to true to prevent client VMs from dynamically downloading classes from other codebases.

13. Briefly explain how remote objects are garbage collected.

Answer:

The RMI garbage collector is based on reference count. A reference count has an associated lease. A lease has a time period for which it is valid. When a remote client (including an RMI registry) gets a reference to a remote object, it sends a message to the RMI runtime on the server requesting a lease for that remote object reference. The server grants a lease for a specified time period to that client. The server increments the reference count for that remote object by one and sends back the lease to the client.

A client keeps renewing the lease for a remote object before the lease expires. When a lease for a remote client is renewed for a remote reference, the server keeps track of the lease expiration time and it will not garbage collect the remote object. When a client is done with a remote object reference, it sends a message to the server that it is done with it. The message is sent when the remote reference in the client's JVM is garbage collected. When the server receives a message that a client does not use the remote object anymore, the server decrements the reference count for that remote object by 1.

Commented [KS1]:

When the reference count of a remote object in the server goes down to zero, the RMI runtime will reference the remote object using a *weak reference*, so if there is no local reference to the remote object, it may be garbage collected.

14. Describe the steps to get notified when a remote object is no longer being referenced.

Answer:

The RMI runtime can notify you when the reference count of the remote object has gone down to zero. It is important to get this notification if a remote object holds some resources that you would like to free when no remote client is referencing it. To get this notification, you need to implement the `java.rmi.server.Unreferenced` interface in your remote object implementation class. Its declaration is as follows:

```
public interface Unreferenced {
    void unreferenced()
|
```

The `unreferenced()` method of the remote object is called when its remote reference count becomes zero.