

## Chapter 29 Remote Method Invocation

1. To define an interface for a remote object, create a subinterface that extends the Remote interface. Every method defined in the interface must declare to throw RemoteException.
2. The stub and skeleton are for handling communications between the client and the server. The stub resides on the client machine. It contains all the reference information the client needs to know about the server object. When a client invokes a method on the server object, it actually invokes a method that is encapsulated in the stub. The stub is responsible for sending parameters to the server, and for receiving the result from the server and returning it to the client. The skeleton communicates with the stub on the server side. The skeleton receives parameters from the client, passes them to the server for execution, and returns the result to the stub.
3. The stub and the skeleton can be automatically generated from the server implementation using an RMI utility called `rmic`. For example, `rmic StudentServerInterfaceImpl` generates `StudentServerInterfaceImpl_Skel.class` and `StudentServerInterfaceImpl_Stub.class`.
4. The implementation of the remote object interface, referred to as *server implementation*, resides on the server machine and is registered with the RMI registry. A client looks through the RMI registry for the remote object. Once it is located, it can be used in the same way as a local object. To create a rmi registry, simply run `rmiregistry`.
5. To start RMI Registry, type "**start rmiregistry**" at a DOS prompt. By default, the port number 1099 is used by `rmiregistry`. To use a different port number, simply type the command "**start rmiregistry portnumber**" at a DOS prompt.
6. To register a remote object with the rmi registry, create a class with the main method. In the main method, bind the object with the rmi registry.
7. Use the following command to start the server

C:\book>java -Djava.security.policy=policy RegisterWithRMIServer

8. A client looks through the RMI registry for the remote object. Once it is located, it can be used in the same way as a local object.
9. You may modify the security file to grant permissions that circumvent the security restriction.
10. See Section 30.4.
11. When a client invokes a remote method with parameters, passing the parameters is handled by the stub and the skeleton. Let us consider three types of parameters:
  - Primitive data type. A parameter of primitive type, such as char, int, double, or boolean, is passed by value like a local call.
  - Local object type. A parameter of local object type, such as java.lang.String, is also passed by value, but this is completely different from passing an object parameter in a local call. In a local call, an object parameter's reference is passed, which corresponds to the memory address of the object. In a remote call, there is no way to pass the object reference because the address on one machine is meaningless to a different Java VM. Any object can be used as a parameter in a remote call as long as it is serializable. The stub serializes the object parameter and sends it in a stream across the network. The skeleton deserializes the stream into an object.
  - Remote object type. Remote objects are passed differently from local objects. When a client invokes a remote method with a parameter of a remote object type, the stub of the remote object is passed. The server receives the stub and manipulates the parameter through it. See Example 16.3, "The RMI TicTacToe Implementation," for details.
12. The token is assigned by the TicTacToe server. You cannot pass the variable back to its caller in RMI programming.
13. In a traditional client/server system, a client sends a request to a server, and the server processes the request and returns the result back to the client. The server cannot invoke the methods on a client. One of the important benefits of RMI is that it supports *callbacks*, which enable the server to invoke the methods on the client. With the RMI callback feature, you can develop interactive distributed applications.