Java RMI Tutorial for Eclipse IDE

Your Name

November 20, 2024

Contents

Java Remote Method Invocation (RMI)

Java RMI, or Remote Method Invocation, allows a Java program to invoke methods on an object located on a different machine or JVM. It's a way to build distributed applications. In this tutorial, we'll implement an example where the client calls a greeting method and an addition method from a remote server.

1 Introduction

This tutorial introduces students to Java Remote Method Invocation (RMI) using Eclipse IDE. By the end of this guide, students will create, deploy, and test a simple RMI-based application where the client can call two different methods on a remote object.

2 Step 1: Setting Up the Eclipse Project

1. **Open Eclipse IDE.** Create a new Java project by navigating to `File > New > Java Project`.
2. **Name the project.** Enter a name like `RMIExample` and click `Finish`.
3. **Create packages.** In the `src` folder, create two packages:
   - `rmi.server` for server-side code.
   - `rmi.client` for client-side code.

3 Step 2: Defining the Remote Interface

- Define a remote interface that declares the methods accessible to the client. This interface must extend `java.rmi.Remote`.
- **Create the interface:** In the `rmi.server` package, create a new interface named `RemoteService`.

```
package rmi.server;

import java.rmi.Remote;
import java.rmi.RemoteException;

public interface RemoteService extends Remote {
    String sayHello(String name) throws RemoteException;
    int addNumbers(int a, int b) throws RemoteException;
}
```

- **Explanation:** The `RemoteService` interface defines two methods:
  - `sayHello`: Accepts a name and returns a greeting.
  - `addNumbers`: Adds two integers and returns the sum.

4 Step 3: Implementing the Remote Interface

- Implement the remote interface in a class that will act as the remote object.
- **Create the class:** In the `rmi.server` package, create a new class named `RemoteServiceImpl`.

```java
package rmi.server;

import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;

public class RemoteServiceImpl extends UnicastRemoteObject
    implements RemoteService {

    protected RemoteServiceImpl() throws RemoteException {
        super();
    }

    @Override
    public String sayHello(String name) throws
        RemoteException {
        return "Hello,␣" + name + "!";
    }

    @Override
    public int addNumbers(int a, int b) throws
        RemoteException {
        return a + b;
    }
}
```

- **Explanation:** This class implements the remote methods. By extending `UnicastRemoteObject`, it can be used as an RMI object.

5 Step 4: Creating the RMI Server

- The server class will create an instance of `RemoteServiceImpl` and bind it to the RMI registry.
- **Create the server class:** In the `rmi.server` package, create a class named `RMIServer`.

```java
package rmi.server;

import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
```

```java
public class RMIServer {

    public static void main(String[] args) {
        try {
            RemoteService service = new RemoteServiceImpl();
            Registry registry = LocateRegistry.
                createRegistry(1099);                  ✓
            registry.rebind("RemoteService", service);
            System.out.println("Server␣is␣running...");  ✓
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

- **Explanation:** The server binds the `RemoteServiceImpl` instance to the RMI registry, making it accessible to cli~~ents via the~~ name `"RemoteService"`.

6 Step 5: Creating the RMI Client ✓

- The client connects to the RMI registry and invokes methods on the remote object.
- **Create the client class:** In the `rmi.client` package, create a class named `RMIClient`.

```java
package rmi.client;

import rmi.server.RemoteService;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;

public class RMIClient {

    public static void main(String[] args) {
        try {
            Registry registry = LocateRegistry.getRegistry("
                localhost", 1099);
            RemoteService service = (RemoteService) registry
                .lookup("RemoteService");

            System.out.println(service.sayHello("Alice"));
            System.out.println("Sum:␣" + service.addNumbers
                (5, 7));
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

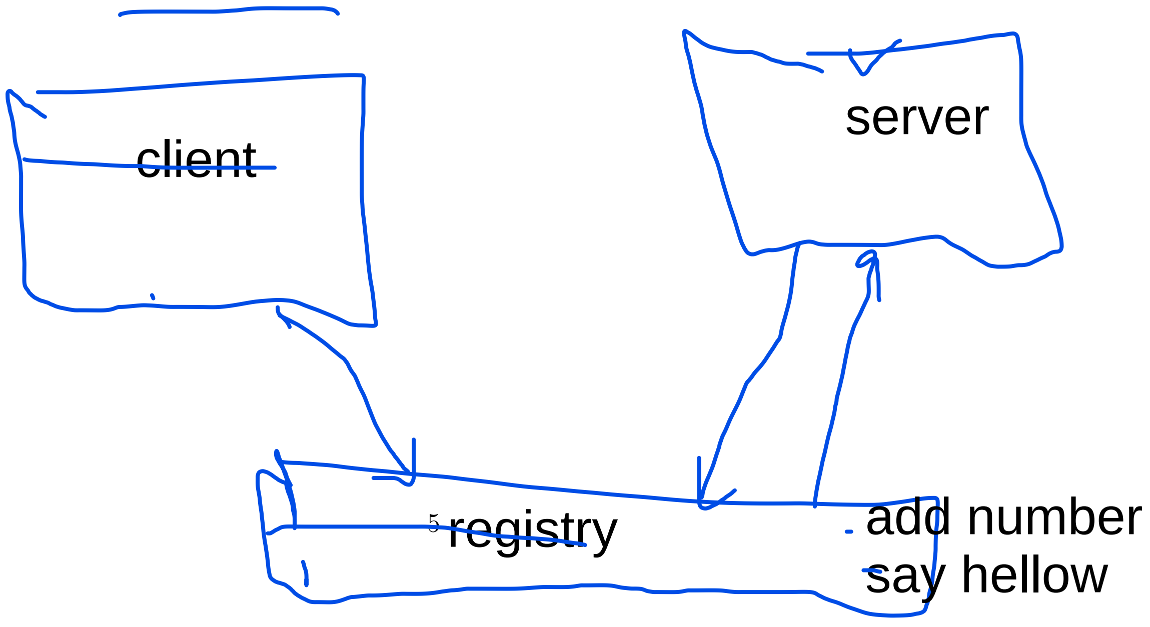
- **Explanation:** The client looks up the `RemoteService` and invokes the `sayHello` and `addNumbers` methods.

7 Step 6: Running and Testing the Application

- Open two separate consoles in Eclipse.
- **Running the server:** Right-click on `RMIServer` and select `Run As > Java Application`.
  - The server console should display `"Server is running..."`.
- **Running the client:** Right-click on `RMIClient` and select `Run As > Java Application`.
  - The client console should display the output of `sayHello` and `addNumbers`.
- **Explanation:** This demonstrates basic RMI usage with two method calls from the client to the remote server.

8 Step 7: Troubleshooting Common Issues

- **RMI Port Issues:** Ensure port 1099 is available on your machine.
- **Firewall Issues:** Temporarily disable the firewall if facing connection issues.
- **ClassNotFoundException:** Check that both client and server share the same `RemoteService` interface.


client
server
registry
add number
say hellow