Aula 04
Sistemas Distribuídos

5º Semestre

ADS

Este texto foi retirado integralmente de um Artigo publicado no site DEVMEDIA:

https://www.devmedia.com.br/uma-introducao-ao-rmi-em-java/28681

O que é RMI?

Java RMI é um mecanismo para permitir a invocação de métodos que residem em diferentes máquinas virtuais Java (JVM). O JVM pode estar em diferentes máquinas ou podem estar na mesma máquina. Em ambos os casos, o método pode ser executado em um endereço diferente do processo de chamada. Java RMI é um mecanismo de chamada de procedimento remoto orientada a objetos.

Aplicação de objetos distribuídos

Uma aplicação RMI é frequentemente composto por dois programas diferentes, um servidor e um cliente. O servidor cria objetos remotos e faz referências a esses objetos disponíveis. Em seguida, ele é válido para clientes invocarem seus métodos sobre os objetos.
O cliente executa referências remotas aos objetos remotos no servidor e invoca métodos nesses objetos remotos.

O modelo de RMI fornece uma aplicação de objetos distribuídos para o programador. Ele é um mecanismo de comunicação entre o servidor e o cliente para se comunicarem e transmitirem informações entre si.

A aplicação de objetos distribuídos tem de prover as seguintes propriedades:
•Localização de objetos remotos: O sistema tem de obter referências a objetos remotos. Isto pode ser feito de duas maneiras. Ou, usando as instalações de nomeação do RMI, o registro RMI, ou passando e retornando objetos remotos.
•Comunicação com objetos remotos: O desenvolvedor não tem de lidar com a comunicação entre os objetos remotos desde que este é tratado pelo sistema RMI.
•Carregar os bytecodes de classe dos objetos que são transferidos como argumentos ou valores.

Todos os mecanismos para carregar o código de um objeto e transmissão dos dados são fornecidos pela RMI system.40

Interfaces e Classes

Java RMI é um sistema de linguagem individual, a programação de aplicação distribuída em RMI é bastante simples. Todas as interfaces e classes para o sistema de RMI são definidos no pacote java.rmi. A classe de objeto remoto implementa a interface remota, enquanto as outras classes estendem RemoteObject.

A interface remota

Uma interface remota é definida pela extensão da interface Remote que está no pacote java.rmi. A interface que declara os métodos que os clientes podem invocar a partir de uma máquina virtual remoto é conhecido como interface remota. A interface remota deve satisfazer as seguintes condições:

•Deve estender-se a interface Remote.

•Cada declaração de método na interface remota deve incluir a exceção RemoteException (ou uma de suas superclasses), em sua cláusula lançada.

A classe RemoteObject

Funções do servidor RMI são fornecidos pela classe RemoteObject e suas subclasses Remote Server, Activatable e UnicastRemoteObject.

Aqui está uma breve descrição de como lidar com as diferentes classes:

•RemoteObject fornece implementações dos métodos toString, equals e hashCode na classe java.lang.Object.

•As classes UnicastRemoteObject e Activatable cria objetos remotos e os exporta, ou seja, essas classes fazem os objetos remotos usados por clientes remotos.

A classe RemoteException

A classe RemoteException é uma super-classe das exceções que o sistema RMI joga durante uma invocação de método remoto. Cada método remoto que é declarado em uma interface remota deve especificar RemoteException (ou uma de suas superclasses), em sua cláusula throws para garantir a robustez das aplicações no sistema RMI.

Quando uma chamada de método remoto tiver um erro, a exceção RemoteException é lançada. Falha de comunicação, erros de protocolo e falha durante a triagem ou unmarshalling de parâmetros ou valores de retorno são algumas das razões para o fracasso da comunicação RMI. RemoteException é uma exceção que deve ser tratada pelo método chamador. O compilador confirma que o programador de ter lidado com essas exceções.

Implementação de um simples sistema RMI

Este é um sistema de RMI simples com um cliente e um servidor. O servidor contém um único método hello (Olá), que retorna uma String para o cliente.

Para criar o sistema RMI todos os arquivos tem que estar compilados. Em seguida, o outline e stub, que são mecanismos de comunicação padrão com objetos remotos, são criadas com o compilador RMIC.

Este sistema RMI contém os seguintes arquivos:

- Hello.java: A interface remota.
- HelloClient.java: A aplicação cliente no sistema RMI.
- HelloServer.java: O aplicativo de servidor no sistema RMI.
- Quando todos os arquivos são compilados, executando o seguinte comando irá criar o stub e o skeleton: RMIc HelloServer

Em seguida, as duas classes serão criados, HelloServer_Stub.class e HelloServer_Skel.class, onde a primeira classe representa o lado do cliente do sistema RMI e o segundo arquivo representa o lado do servidor do sistema RMI.

Listagem 1: Hello.java

```
import java.RMI.Remote;
import java.RMI.RemoteException;
/*
Classname: Hello
Comment: The remote interface.
*/
public interface Hello extends Remote {
        String Hello() throws RemoteException;
}
```

Listagem 2: HelloClient.java

```
import java.RMI.Naming;
import java.RMI.RemoteException;

/*
Classname: HelloClient
Comment: The RMI client.
*/
public class HelloClient {
    static String message = "blank";

public static void main(String[] args) {
      RMI_Hello_Client client = new
RMI_Hello_Client();
      client.connectServer();
   }
```

```
  public void connectServer() {
      Registry registry;

      try {
         registry =
LocateRegistry.getRegistry("192.168.1.139", 1099);
         RMI_Hello rmi = (RMI_Hello)
registry.lookup("HelloServer");
         System.out.println("Connect to Server");
         String text = rmi.Hello();
         System.out.println(text);
      } catch (RemoteException ex) {

Logger.getLogger(RMI_Hello_Client.class.getName(
)).log(Level.SEVERE, null, ex);
      } catch (NotBoundException ex) {

Logger.getLogger(RMI_Hello_Client.class.getName(
)).log(Level.SEVERE, null, ex);
      }
   }
```

Listagem 3: HelloServer.java

```
import java.RMI.Naming;
import java.RMI.RemoteException;
import java.RMI.RMISecurityManager;
import java.RMI.server.UnicastRemoteObject;
/*
Classname: HelloServerDemo
Purpose: The RMI server.
*/
public class HelloServerDemo extends      UnicastRemoteObject
implements Hello {
public HelloServerDemo() throws RemoteException {
super();
}
public String Hello() {
System.out.println("Invocation to Hello was succesful!");
return "Hello World from RMI server!";
}
```

```
public static void main(String[] args) {
       // CreatesanobjectoftheHelloServerclass.
        Registry registry;
        try {
           registry =
LocateRegistry.createRegistry(1099);
           registry.rebind("HelloServer", new
RMI_Hello_Server());
           System.out.println("Server started");
       } catch (RemoteException ex) {

Logger.getLogger(RMI_Hello_Server.class.getName()
).log(Level.SEVERE, null, ex);
       }
    }
```

Saída:

Mensagem do servidor RMI de: Joe

Ligado no registro

Listagem 4: Programa para implementar Calculadora usando RMI-Calcutator.java

```
import java.RMI.Remote;
import java.RMI.RemoteException;

public interface Calcutator extends Remote
{
    public long add(long a, long b) throws RemoteException;
}
```

Listagem 5: CalculatorImple.java

```
import java.RMI.RemoteException;
import java.RMI.server.UnicastRemoteObject;

public class CalculatorImple extends UnicastRemoteObject implements Calculator
{
    protected CalculatorImple() throws RemoteException
    {
        super();
    }
    public long add(long a, long b) throws RemoteException
    {
        return a+b;
    }
}
```

Listagem 6: CalculatorServer.java

```
import java.RMI.Naming;

public class CalculatorServer
{
    CalculatorServer()
    {
        try
        {
            Calcutator c = new CalculatorImple();
            Naming.rebind("RMI://127.0.0.1:1020/CalculatorService",  c);
        }
        catch (Exception e)
        {
            e.printStackTrace();
        }
    }
    public static void main(String[] args)
    {
        new CalculatorServer();
    }
}
```

Listagem 7: CalculatorClient.java

```
import java.RMI.Naming;
public class CalculatorClient
{
    public static void main(String[] args)
    {
        try
        {
            Calculator c = (Calculator) Naming.lookup("//127.0.0.1:1020/CalculatorService");
            System.out.println("Adição : "+c.add(20, 15));
        }
        catch (Exception e)
        {
            System.out.println(e);
        }
    }
}
```

Conclusão

Neste artigo, vimos como podemos usar diferentes métodos em diferentes JVM e combiná-los para usar como um único aplicativo. RMI é útil em um aplicativo do servidor onde se quer comunicar com diferentes JVM.RMI usa serialização marshal e unmarshal, reforçando verdadeiro polimorfismo orientado a objetos.

Artigo traduzido e originalmente publicado em: ***http://mrbool.com/rmi-remote-method-invocation-in-java/28575***