Università degli Studi di Padova

Il modello Java RMI

SCD

Anno accademico 2004/5
Corso di Sistemi Concorrenti e Distribuiti

Tullio Vardanega, tullio.vardanega@math.unipd.it

Architettura del modello - 1

- **Oggetto remoto come sola forma di oggetto distribuito**
  - **L'interfaccia può essere resa accessibile a processi remoti**
  - **Lo stato risiede sempre su un singolo nodo**
- **Oggetto remoto ≠ oggetto locale 1/2**
  - **Rispetto alla clonazione**
    - Solo il suo servente può clonare un oggetto remoto, che viene creato nello spazio di indirizzamento del servente
    - I *proxy* dell'oggetto remoto originale non vengono clonati → il cliente che volesse utilizzare il clone deve localizzarlo e connettersi esplicitamente

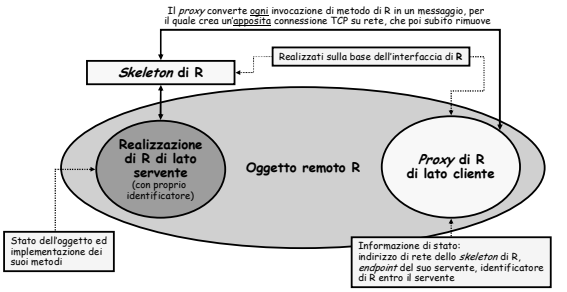
Architettura del modello - 2

- **Oggetto remoto ≠ oggetto locale 2/2**
  - **Rispetto alla mutua esclusione**
    - La presenza di metodi **`synchronized`** su un oggetto remoto non garantisce mutua esclusione tra processi clienti che risiedano su nodi distinti
    - Ogni *proxy* di oggetto remoto garantisce infatti mutua esclusione solo ai processi che risiedano sullo stesso nodo del *proxy*
  - **Rispetto ai parametri passati ai metodi**
    - Il tipo dell'oggetto passato come parametro ad RMI deve essere idoneo a *marshalling* ed *unmarshalling* → parametro di tipo **`serializable`**
      - Tutti i tipi primitivi lo sono tranne quelli che dipendono dalla specifica JVM (p.es. Thread, descrittori di *file, socket*) e sono "insicuri" (p.es. FileInputStream)
  - **Rispetto a come l'oggetto possa essere passato come parametri**
    - Oggetto locale → per valore
    - Oggetto remoto → per riferimento

Architettura del modello - 3

- **Per il resto, oggetto remoto ed oggetto locale sono indistinguibili**
- **Riferimento all'oggetto remoto**
  - **Indirizzo di rete (IP), *endpoint* del servente, modalità di comunicazione (*protocol stack*)**
    - Usato dal *proxy* (chiamato *stub*)
  - **Identificatore locale dell'oggetto nello spazio del servente**
    - Usato dal servente

Architettura del modello - 4



Architettura del modello - 5

- **Il *proxy* è serializzabile**
  - **Può essere passato come parametro (per valore) ed usato dal ricevente come riferimento all'oggetto remoto**
  - **Poiché tutto esegue su JVM standard non occorre copiare il codice del *proxy* presso il cliente → basta indicare le classi che occorrono per rigenerarlo a destinazione**
    - Mobilità **forte** rispetto allo stato
    - Legame **debole** rispetto al codice come risorsa (*binding by value*)
- **L'uso di RMI pertanto consente migrazione di copia del *proxy* verso il chiamante**

Università degli Studi di Padova

Sistemi distribuiti: il modello Java RMI

Utilizzo del modello - 1

- **L'oggetto remoto deriva da una interfaccia pubblica che estende `java.rmi.Remote`**

```
import java.rmi.*;
public interface Echo extends Remote {
  String call (String message) throws RemoteException;}
```

- **Ogni suo metodo può emettere eccezione `java.rmi.RemoteException`**
- **Ogni uso dell'oggetto come argomento o valore di ritorno ha il tipo dell'interfaccia e non della sua implementazione**

Università degli Studi di Padova

Sistemi distribuiti: il modello Java RMI

Utilizzo del modello - 2

- **Il servente dell'oggetto remoto deve**
  - **Estendere `java.rmi.UnicastRemoteObject`**
  - **Implementare i metodi dell'oggetto remoto**
  - **Definire esplicitamente un costruttore dell'oggetto che possa emettere eccezione `java.rmi.RemoteException`**

```
import java.rmi.*;
import java.rmi.server.*
public class EchoServer extends UnicastRemoteObject implements Echo{
  public EchoServer( String name ) throws RemoteException {
   Naming.rebind (name, this); }
  public String call (String message) throws RemoteException {
   return "From EchoServer:- message: [" + message + "]"; }
  public static void main (String args[]) {
   // il main è nel servente, che può anche essere separato dalla
   // classe che implementa l'oggetto remoto }
```


Università degli Studi di Padova

Sistemi distribuiti: il modello Java RMI

Utilizzo del modello - 3

- **La logica del servente è specificata nel suo `main`, che crea istanze dell'oggetto remoto**
- **Ogni istanza deve essere registrata presso il registro degli oggetti remoti del nodo, che viene mantenuto da un processo dedicato (`rmiregistry`)**
  - `Naming.bind` — **lega un nome (stringa URL) all'oggetto remoto (al suo riferimento) in una associazione unica e non modificabile**
  - `Naming.rebind` — **crea una nuova associazione (nome, riferimento) anche sovrascrivendo quella precedente**
- **Il gestore del registro (*name server* locale) ascolta su una porta assegnata (*default*: 1099)**

Università degli Studi di Padova

Sistemi distribuiti: il modello Java RMI

Utilizzo del modello - 4

- **Il *name server* deve essere attivato a parte**
  - `start rmiregistry [portnumber]` ← **Win32**
  - `rmiregistry [portnumber] &` ← **Linux**
- **Una singola istanza di *name server* opera per conto di tutti i serventi di oggetti remoti del nodo**
- **Le varie componenti (interfaccia, oggetto remoto, servente, cliente) vengono compilate in 2 fasi distinte → `javac` ed `rmic`**

Università degli Studi di Padova

Sistemi distribuiti: il modello Java RMI

Utilizzo del modello - 5

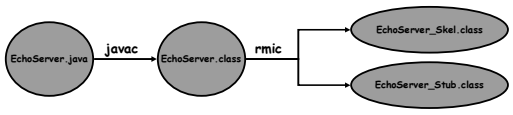
- **Il cliente dell'oggetto remoto deve**
  - **Localizzarne (tramite *look-up*) il riferimento presso il registro dei nomi**
    - `Naming.lookup (String name)`
      - Dove *name* è l'URL della specifica dell'oggetto remoto (la sua interfaccia pubblica) presso il nodo e la porta dove è in ascolto il corrispondente gestore dei nomi
  - **Il riferimento così ottenuto ha il tipo dell'interfaccia e non della classe che lo realizza!**
- **D'ora in poi l'oggetto remoto è del tutto indistinguibile da un oggetto locale**

Università degli Studi di Padova

Sistemi distribuiti: il modello Java RMI

Utilizzo del modello - 6

- **`rmic` (compilatore Java RMI)** 1/2
  - **Genera *stub* e *skeleton* per oggetti remoti a partire dalle classi compilate che ne contengono l'implementazione**
  - **Le classi compilate di partenza devono essere identificate rispetto ai *package* che le contengono**



Università degli Studi di Padova

Sistemi distribuiti: il modello Java RMI

Utilizzo del modello - 7

- `rmic` (compilatore Java RMI) 2/2
  - Lo *skeleton* è una entità di lato servente che contiene un metodo che recepisce le chiamate remote all'oggetto e le indirizza verso la sua implementazione
    - Il protocollo utilizzato è specifico di Java RMI (**JRMP**)
  - Lo *stub* è il *proxy* dell'oggetto remoto, che indirizza le chiamate ad esso verso il servente corrispondente
    - Il riferimento all'oggetto remoto in possesso del cliente riferisce in realtà lo *stub* dell'oggetto, ossia il *proxy* locale al cliente
    - Lo *stub* di lato servente riproduce le chiamate remote in ingresso e le gira localmente allo *skeleton* corrispondente

Università degli Studi di Padova

Sistemi distribuiti: il modello Java RMI

Applicazione del modello - 1

- La JVM consente di caricare dinamicamente *software* (*bytecode*) Java da qualsiasi URL
  - Capacità utilizzabile da RMI
- Le classi locali vengono normalmente caricate a partire dalla locazione `CLASSPATH`
- Le classi remote possono essere caricate a partire dall'URL `codebase`
  - Locazione configurata come proprietà
    `java -Djava.rmi.server.codebase=file://<path>/`

Università degli Studi di Padova

Sistemi distribuiti: il modello Java RMI

Applicazione del modello - 2

- L'accesso a classi sconosciute può essere regolato da un gestore della sicurezza
  - Lato servente → consentire la copia di proprie classi
  - Lato cliente → impedire l'accesso a siti non affidabili
- Accesso regolamentato da specifica politica configurata in un *file* passato come proprietà
  - `java -Djava.security.policy = <policy_file>`

```
grant {
 permission java.io.FilePermission "<<ALL FILES>>", "read";
 permission java.net.SocketPermission "*:1234", "accept, connect, listen, resolve";
 permission java.lang.RuntimePermission "accessClassInPackage.sun.jdbc.odbc";
 permission java.util.PropertyPermission "file.encoding", "read"; };
```


Università degli Studi di Padova

Sistemi distribuiti: il modello Java RMI

Esempio - 1

```
package echo;
public interface Echo extends java.rmi.Remote {
  String call (String message) throws java.rmi.RemoteException;
}
```

```
package echo; import java.rmi.*; import java.rmi.server.*;
public class EchoServer extends UnicastRemoteObject implements Echo {
   public EchoServer( String name ) throws RemoteException {
      try { Naming.rebind (name,this); } catch (Exception e) {
         System.out.println ("Exception in EchoServer: " + e.getMessage());
         e.printStackTrace();} }
   public String call (String message) throws RemoteException {
      System.out.println("Echo's method call invoked: [" + message + "]");
      return "From EchoServer:- Thanks for your message: [" + message + "]"; }
   public static void main (String args[]) throws Exception {
      if (System.getSecurityManager() == null)
         System.setSecurityManager ( new RMISecurityManager() );
      String url = "rmi://" + args[0] + "/Echo";
      EchoServer echo = new EchoServer (url);
      System.out.println("EchoServer ready!"); }
}
```


Università degli Studi di Padova

Sistemi distribuiti: il modello Java RMI

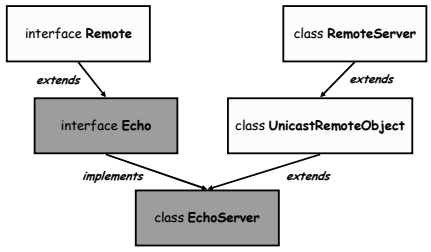
Esempio - 2

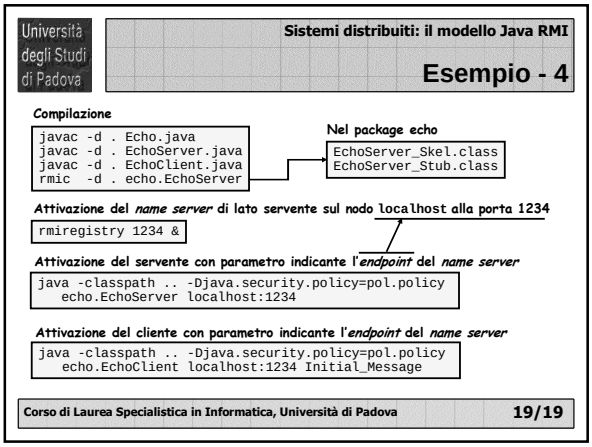
```
package echo; import java.rmi.*; import java.rmi.server.*;
public class EchoClient {
   public static void main (String args[]) {
      int i;
      if (System.getSecurityManager() == null)
         System.setSecurityManager ( new RMISecurityManager() );
      try {
         System.out.println ("EchoClient ready!");
         String url = "rmi://" + args[0] + "/Echo";
         System.out.println ("Looking up remote object " + url + " ...");
         Echo echo = (Echo) Naming.lookup (url);
         String toMsg = (String) args[1];
         for (i = 1; i<6; i++) {
            toMsg = toMsg + "-" + i;
            System.out.println ("Message " + i + " to Echo: [" + toMsg + "]");
            String fromMsg = echo.call (toMsg);
            Thread.sleep (2000);
            System.out.println ("Message from Echo: \n\t" + fromMsg + "\n"); }
      } catch (Exception e) {
         System.out.println ("Exception in EchoClient: " + e.getMessage());
         e.printStackTrace(); } }
}
```


Università degli Studi di Padova

Sistemi distribuiti: il modello Java RMI

Esempio - 3




Università degli Studi di Padova
Sistemi distribuiti: il modello Java RMI
Esempio - 4
Compilazione
javac -d . Echo.java
javac -d . EchoServer.java
javac -d . EchoClient.java
rmic -d . echo.EchoServer
Nel package echo
EchoServer_Skel.class
EchoServer_Stub.class
Attivazione del name server di lato servente sul nodo localhost alla porta 1234
rmiregistry 1234 &
Attivazione del servente con parametro indicante l'endpoint del name server
java -classpath .. -Djava.security.policy=pol.policy
echo.EchoServer localhost:1234
Attivazione del cliente con parametro indicante l'endpoint del name server
java -classpath .. -Djava.security.policy=pol.policy
echo.EchoClient localhost:1234 Initial_Message
Corso di Laurea Specialistica in Informatica, Università di Padova
19/19