

Università degli Studi di Padova

Il modello Java RMI

SCD

Anno accademico 2003/4
Corso di Sistemi Concorrenti e Distribuiti

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

Corso di Laurea Specialistica in Informatica, Università di Padova 1

Università degli Studi di Padova

Sistemi distribuiti: il modello Java RMI

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 $\neq$ oggetto locale 1/2**
 - **Rispetto alla clonazione**
 - Solo il suo servente può clonare un oggetto remoto, che viene creato nel suo spazio di indirizzamento
 - I proxy dell'oggetto remoto originale non vengono clonati $\rightarrow$ il cliente che volesse utilizzare il clone deve localizzarlo e connettersi esplicitamente

Corso di Laurea Specialistica in Informatica, Università di Padova 2

Università degli Studi di Padova

Sistemi distribuiti: il modello Java RMI

Architettura del modello - 2

- ❑ **Oggetto remoto $\neq$ 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 risiedono su nodi distinti
 - Ogni *proxy* di oggetto remoto garantisce infatti mutua esclusione solo ai processi che risiedono sul suo stesso nodo (di quello del *proxy*)
 - **Rispetto ai parametri passati ai metodi**
 - Il tipo dell'oggetto passato come parametro ad RMI deve essere idoneo a *marshalling* ed *unmarshalling* $\rightarrow$ *serializable*
 - Tutti tranne i tipi con dipendenze dalla piattaforma di esecuzione (p.es. descrittori di file, socket)
 - **Rispetto a come loro possano essere passati come parametri**
 - Oggetto locale $\rightarrow$ per valore
 - Oggetto remoto $\rightarrow$ per riferimento

Corso di Laurea Specialistica in Informatica, Università di Padova 3

Università degli Studi di Padova

Sistemi distribuiti: il modello Java RMI

Architettura del modello - 3

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

Corso di Laurea Specialistica in Informatica, Università di Padova 4

Università degli Studi di Padova

Sistemi distribuiti: il modello Java RMI

Architettura del modello - 4

Il *proxy* converte ogni invocazione di metodo di R in un messaggio, per il quale crea un'apposita connessione su rete, che poi subito rimuove

Corso di Laurea Specialistica in Informatica, Università di Padova 5

Università degli Studi di Padova

Sistemi distribuiti: il modello Java RMI

Architettura del modello - 5

- ❑ **Il *proxy* è serializzabile**
 - Può essere passato come parametro (per valore) ed usato dal ricevente come riferimento al suo oggetto remoto
 - Poiché tutto esegue su JVM standard non occorre copiare il codice del *proxy* presso il cliente $\rightarrow$ 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**

Corso di Laurea Specialistica in Informatica, Università di Padova 6

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

Corso di Laurea Specialistica in Informatica, Università di Padova

7

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 + " ";
    }
    public static void main (String args[]) {
        // il main è nel servente, che può anche essere separato dalla
        // classe che implementa l'oggetto remoto
    }
}

```

Corso di Laurea Specialistica in Informatica, Università di Padova

8

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)

Corso di Laurea Specialistica in Informatica, Università di Padova

9

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
- ❑ Potendo così operare 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`

Corso di Laurea Specialistica in Informatica, Università di Padova

10

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

Corso di Laurea Specialistica in Informatica, Università di Padova

11

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

```
graph LR
    A(EchoServer.java) -- javac --> B(EchoServer.class)
    B -- rmic --> C(EchoServer_Skel.class)
    B -- rmic --> D(EchoServer_Stub.class)

```

Corso di Laurea Specialistica in Informatica, Università di Padova

12

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

Corso di Laurea Specialistica in Informatica, Università di Padova

13

Università degli Studi di Padova

Sistemi distribuiti: il modello Java RMI

Applicazione del modello - 1

❑ La JVM consente di caricare dinamicamente **software 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>/

Corso di Laurea Specialistica in Informatica, Università di Padova

14

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";
}
```

Corso di Laurea Specialistica in Informatica, Università di Padova

15

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 + "]\n");
        return "From EchoServer- Thanks for your message: [" + message + "]\n";
    }
    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!");
    }
}
```

Corso di Laurea Specialistica in Informatica, Università di Padova

16

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 + "]\n");
                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();
        }
    }
}
```

Corso di Laurea Specialistica in Informatica, Università di Padova

17

Università degli Studi di Padova

Sistemi distribuiti: il modello Java RMI

Esempio - 3

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

18