Remote Method Invocation (RMI)

Remote Method Invocation (RMI) in Java.

Walter Cazzola
Dipartimento di Informatica e Comunicazione
Università degli Studi di Milano.

e-mail: cazzola@disi.unige.it

Remote Method Invocation (RMI)

RMI permette ad un oggetto di attivare metodi di un altro oggetto (remoto) indipendentemente da dove si trovi (cioè appartenenti a RTS distinti).

I dettagli relativi alla comunicazione distribuita scompaiono. Tutti metodi sono invocati seguendo i dettami delle invocazioni classiche.

Remote Method Invocation (RMI)

Le classi che intendono rendere disponibili dei metodi per invocazioni remote devono implementare un'interfaccia che li dichiara.

Stub e Skeleton devono essere creati per interfacciare i due oggetti. In Java, sono creati automaticamente da *rmic*.

- lo stub traduce ogni invocazione del client in una comunicazione destinata al server.
- lo skeleton è il corrispondente dello stub sul lato server e accetta le comunicazioni dal client e le traduce in invocazioni.

Remote Method Invocation
(registrazione sul name server)

L'oggetto remoto per poter essere localizzato deve essere registrato in un name server (rmiregistry).

Il client contatterà il name server per avere un riferimento all'oggetto remoto.

Il client si aspetta un rappresentante del server tramite il quale potrà richiedergli l'esecuzione del servizio.

Remote Method Invocation
(registrazione sul name server)

L'identificazione è fatta tramite il nome con cui l'oggetto remoto si è registrato. Il name server ritornerà il riferimento corrispondente.

Il riferimento contiene:

- L'host su cui è in esecuzione
- La porta su cui è in attesa, e
- OID

Tutte informazioni usate internamente. Per il programmatore il riferimento è uno stub che implementa i servizi richiesti.

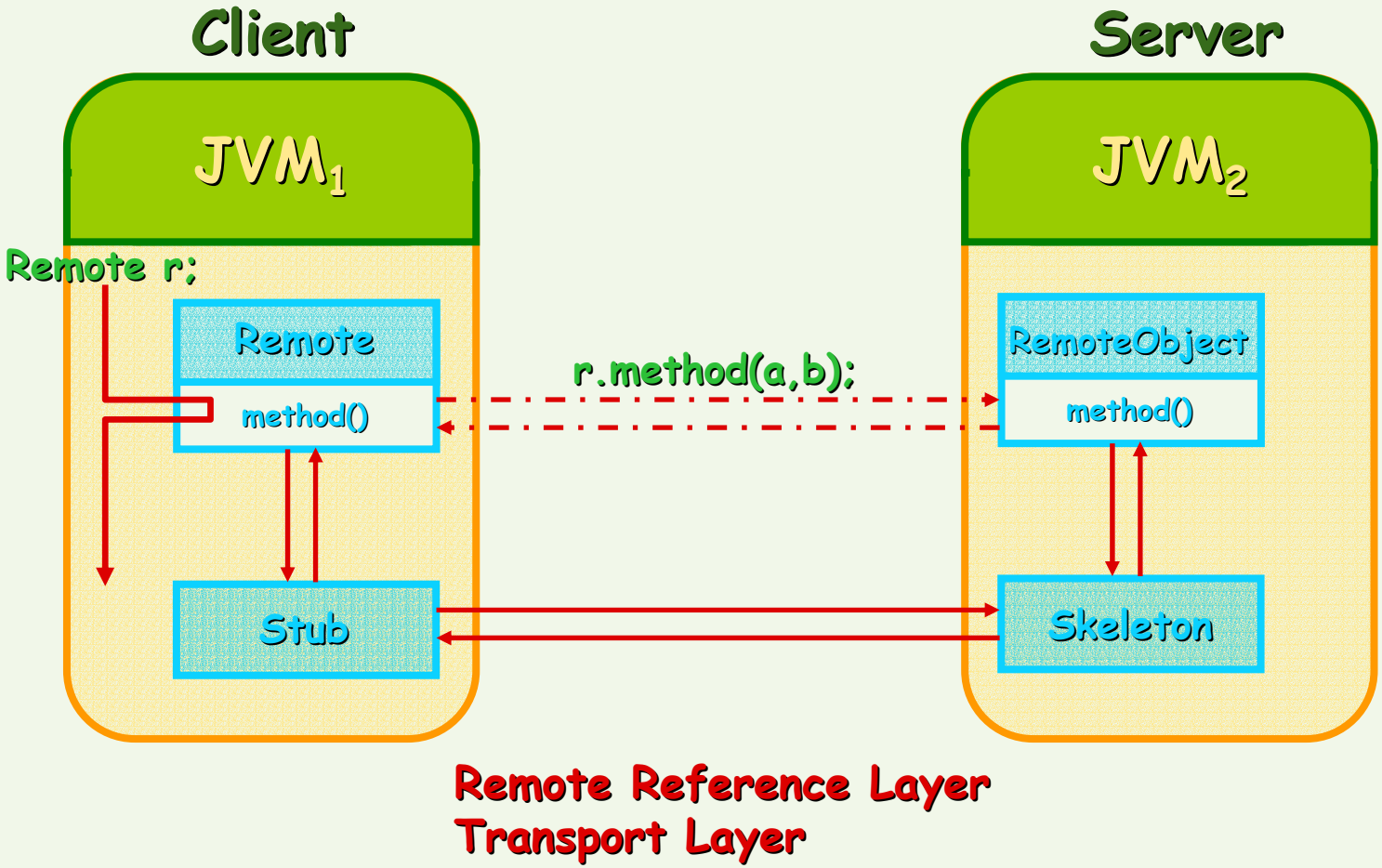
Remote Method Invocation
(invocazione)

Ottenuto il riferimento remoto, il cliente può procedere a invocare metodi dell'oggetto remoto, come se fossero metodi locali. Tutte le comunicazioni di basso livello sono realizzate dal framework sottostante.

Il framework supporta anche un sistema di eccezioni remote e di garbage collection remota.

La chiamata viene intercettata dallo stub che la traduce e la invia allo skeleton del destinatario che la ritraduce ed attiva il metodo in questione. Infine lo skeleton restituirà i valori calcolati allo stub che li ritornerà al cliente come se fosse una chiamata locale.

Remote Method Invocation
(schema di invocazione remota)



Remote Method Invocation
(oggetti remoti: interfaccia)

Un oggetto remoto accetta richieste da oggetti in esecuzione su altre java virtual machine.

L'oggetto remoto deve avere un'interfaccia che definisce quali sono i suoi metodi pubblici.

L'interfaccia deve estendere: java.rmi.Remote ed i metodi sollevare l'eccezione java.rmi.RemoteException.

MyRemote.java

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

public interface MyRemote extends Remote {
  public int remoteHash(String s) throws RemoteException;
}
```

Remote Method Invocation
(oggetti remoti: implementazione)

L'implementazione dell'oggetto remoto deve estendere la classe: java.rmi.server.RemoteObject o una sua sottoclasse, ed implementare tutte le interfacce remote che vuole supportare.

MyRemoteImpl.java

```
import java.rmi.*;
import java.rmi.server.*;

public class MyRemoteImpl extends RemoteObject implements MyRemote {
    public MyRemoteImpl() throws RemoteException {}
    public int remoteHash(String s) {
        return s.hashCode();
    }
}
```

Remote Method Invocation
(stub & skeleton)

Generazione di stub & skeleton

Vengono generati automaticamente a partire dall'implementazione dell'oggetto remoto.

```
rmic MyRemoteImpl
```

Risultato: MyRemoteImpl_Stub.class, e MyRemoteImpl_Skel.class.

Operato di stub & skeleton

stub e skeleton contengono il codice relativo alla serializzazione dei dettagli dell'invocazione e dei parametri di ritorno.

Si basano sulle classi **ObjectOutputStream** e **ObjectInputStream**.

Remote Method Invocation
(registrare e riferire l'oggetto remoto)

Il name server

- Per poter invocare metodi remoti, i clienti devono avere un riferimento all'oggetto.
- Fornire i riferimenti è compito del name server (rmiregistry).

Registrarsi

- gli oggetti remoti si possono registrare tramite la classe: **java.rmi.Naming** ed adotteranno uno schema basato sugli URL per identificarsi.

```
MyRemoteImpl remote-object = new MyRemoteImpl();
Naming.bind("Oggetto Remoto", remote-object);
```

Recupero riferimenti

- Il recupero dei riferimetni viene fatto tramite la classe: **java.rmi.Naming**.

```
MyRemote ref = (MyRemote) Naming.lookup("RMI://"+args[0]+"/Oggetto Remoto");
```

Remote Method Invocation
(esempio data server: RMI server side)

DateServer.java

```
public interface DateServer extends Remote {
   public Date getDate() throws RemoteException;
}
```

```
public class DateServerImpl extends UnicastRemoteObject
                            implements DateServer {
  public DateServerImpl() throws RemoteException {}
  public Date getDate() { return new Date(); }
  public static void main(String[] args) throws Exception {
     DateServerImpl dateServer = new DateServerImpl();
     Naming.bind("Date Server", dateServer);
  }
}
```

DateServerImpl.java

Remote Method Invocation
(esempio data server: RMI client side)

```
public class DateClient {
  public static void main(String[] args) throws Exception {
    if (args.length != 1)
      throw new IllegalArgumentException("Syntax: DateClient <hostname>");
    DateServer dateServer = (DateServer)Naming.lookup("RMI://"+args[0]+"/Date Server");
    Date when = dateServer.getDate();
    System.out.println(when);
  }
}
```

DateClient.java

Remote Method Invocation
(download classi remote)

Si hanno 5 tipologie di classi coinvolte in un'invocazione di metodo remoto:

1. Implementazione del cliente
2. Implementazione del server
3. Interfaccia dell'oggetto remoto
4. Stub
5. Skeleton

Per l'esecuzione del cliente sono necessarie le classi: 1, 3 e 4.
Invece per l'esecuzione del server occorrono le classi: 2, 3, 4 e 5.

Nota una simile distribuzione vale nel caso generale, ma RMI permette oltre a computazioni client-server anche computazioni peer to peer.

Remote Method Invocation
(download classi remote)

Il cliente deve poter accedere allo stub del server. Sia client che server devono avere a disposizione le classi degli oggetti che si scambiano come parametri. Spesso questo non è possibile.

Svantaggio: poco flessibile.

Le classi non riconosciute sono caricate da locazioni predefinite

- CLASSPATH locale,
- Se sono parametri di un RMI, si usano le informazioni usate durante la registrazione,
- Proprietà di sistema **java.rmi.server.codebase**.

Problemi dal punto di vista della sicurezza: Java caricherà classi da server remoti solo se questi avranno installato un **SecurityManager**.

Remote Method Invocation
(interfaccia Remote)

Solo i metodi definiti in un'interfaccia che estende Remote possono essere invocati tramite RMI.

Il passaggio dei parametri è per valore (side effect sugli oggetti).

Tutte le classi coinvolte in un'interfaccia devono essere serializzabili.

Tutti metodi devono dichiarare la possibilità di sollevare l'eccezione RemoteException

```
public interface BankAccount extends Remote {
  public void deposit(int amount) throws RemoteException;
  public void withdraw(int amount) throws RemoteException, InsufficientFundsException;
  public int balance() throws RemoteException;
  public Portfolio addToPortfolio(Portfolio portfolio) throws RemoteException;
}
```

Remote Method Invocation
(classe Naming)

Naming serve per indirizzare gli oggetti remoti:

- i servizi sono identificati tramite uno schema URL-like:

  RMI://hostname:port/service-name

Remote lookup(String address)

- Ritorna un riferimento all'oggetto remoto identificato da address, se address non è un URL corretto viene sollevata l'eccezione **MalformedURLException**, se il servizio richiesto non è registrato viene sollevata l'eccezione **NotBoundException**.

void bind(String address, Remote obj)

- registra l'oggetto remoto obj identificandolo come address, se un oggetto remoto è già stato registrato come address allora viene sollevata l'eccezione **AlreadyBoundException**.

Remote Method Invocation
(classe RemoteObject)

Ogni classe, le cui istanze sono oggetti remoti, DEVE estendere RemoteObject o classi derivate:

- RemoteObject implementa Serializable, per permettere la serializzazione degli oggetti remoti.

Se un metodo remoto prevede un oggetto come parametro, verrà passato un riferimento e non l'oggetto stesso.

- È un errore definire un metodo che attende lo stub (o lo skeleton) di un oggetto remoto.

```
public class MyImpl extends UnicastRemoteObject implements MyRemote {
  public static void main(String[] args) throws Exception {
     MyImpl mine = new MyImpl();
     HisRemote his = (HisRemote) Naming.lookup("his");
     his.invoke(mine);
  }
}
```

devono essere interfacce

Remote Method Invocation
(classe RemoteObject)

Serializzazione

- Un oggetto remoto può essere serializzato su disco come qualsiasi altro oggetto e poi deserializzato altrove.

```
ObjectOutputStream objectOut;
                    ...
MyImpl mine = new MyImpl();
objectOut.writeObject(mine);
ObjectInputStream objectIn;
                    ...
MyImpl mine2 = (MyImpl)objectIn.readObject();
```

Remote Method Invocation
(estensioni di RemoteObject)

RemoteServer

Fornisce le funzioni comuni di un oggetto remoto che ha il ruolo di server. Cioè apre un socket su cui accetta le richieste dei clienti.

UnicastRemoteObject

Salvo casi eccezionali tutti gli oggetti remoti con ruolo di server saranno derivati da questa classe.

- **Clonabile (permette di ottenere un oggetto remoto copia dell'oggetto corrente)**
- **Serializzabile (gli oggetti serializzati sono ancora oggetti remoti)**
- **Accessibile tramite TCP**

Remote Method Invocation
(passaggio dei parametri)

Tipi primitivi

- passati per valore;

Oggetti remoti

- passati per riferimento;

Oggetti non remoti

- passati per valore;
- si usa la Java Object Serialization (Serializzazione).

Remote Method Invocation
(serializzazione)

Serializzazione di un oggetto è sinonimo di persistenza:

- salva lo stato (cioè i dati) di una particolare istanza di un oggetto.

Pertanto:

- *serializzare significa salvare i dati;*
- *deserializzare significa ricaricare i dati.*

Remote Method Invocation
(java.io.Serializable)

Java serializza scrivendo gli oggetti come una sequenza di byte in uno stream e li usa per ricreare un altro oggetto (nuovo) con gli stessi dati.

Gli oggetti che implementano l'interfaccia java.io.Serializable sono serializzabili.

Tramite la reflection, tutti i dati (tranne quelli statici e quelli transienti) saranno serializzati.

Remote Method Invocation
(serializzazione)

Non tutti gli oggetti sono serializzabili:

- Gli oggetti che non implementano Serializable;
- Gli oggetti che rappresentano un rischio per la sicurezza, es. FileInputStream
- Gli oggetti i cui valori dipendono dalla specifica virtual machine, es. Thread;
- Gli oggetti che contengono oggetti non serializzabili (per ricorsione).

Provando a serializzare un oggetto non serializzabile, l'eccezione NotSerializableException viene sollevata.

Remote Method Invocation
(deserializzazione)

Imcompatibilità.

- Se si aggiungono o tolgono campi da una classe, questa diventa *incompatibile* con i dati serializzati precedemente.
- se si tenta di deserializzare oggetti incompatibili l'eccezione java.io.InvalidClassException è sollevata.

Forzare la compatibilità.

- si implementa il metodo readObject() per rendere compatibili i cambiamenti.

```
private void readObject(ObjectInputStream stream) throws IOException {
      defaultReadObject(stream);
      // do compatible stuff
}
```

Remote Method Invocation
(callback)

Callback: funzionano.

- si passa il riferimento remoto al cliente, ed
- il server riesce a chiamare i propri metodi trasparentemente;
- senza interagire con l'RMIRegistry.

RMI Security
(RMI Security Manager)

Sicurezza: i server non sono sicuri e gli stub potrebbero contenere codice malizioso. Serve un SecurityManager.

```
System.setSecurityManager(new RMISecurityManager());
```

AppletSecurityManager

- stub possono fare solo ciò che è consentito agli applet.

RMISecurityManager

- disabilita tutte le funzionalità ad eccezione della definizione e dell'accesso alle classi;
- la classe scaricata può fare una connessione se la connessione è stata iniziata tramite RMI.

Nessuno

- il caricamento degli stub è disabilitato;
- gli stub funzioneranno solo se sono già nel classpath locale.

Remote Method Invocation
(limitazioni)

RMI è solo Java

- si possono usare delle alternative server, es. via JNI.

Usa TCP e non UDP;

Vengono creati almeno due socket per connessione.

Riferimenti Bibliografici

Java Network Programming -Hughes et al. - Cap. 23 e 24
Java Remote Method Invocation Specification. 1998