Remote Method Invocation (RMI)

Lo scopo del RMI consiste nel fornire uno strumento che permette ad un oggetto di attivare metodi di un altro oggetto <u>indipendentemente da dove si trovano</u> (cioè appartenenti a RTS distinti).

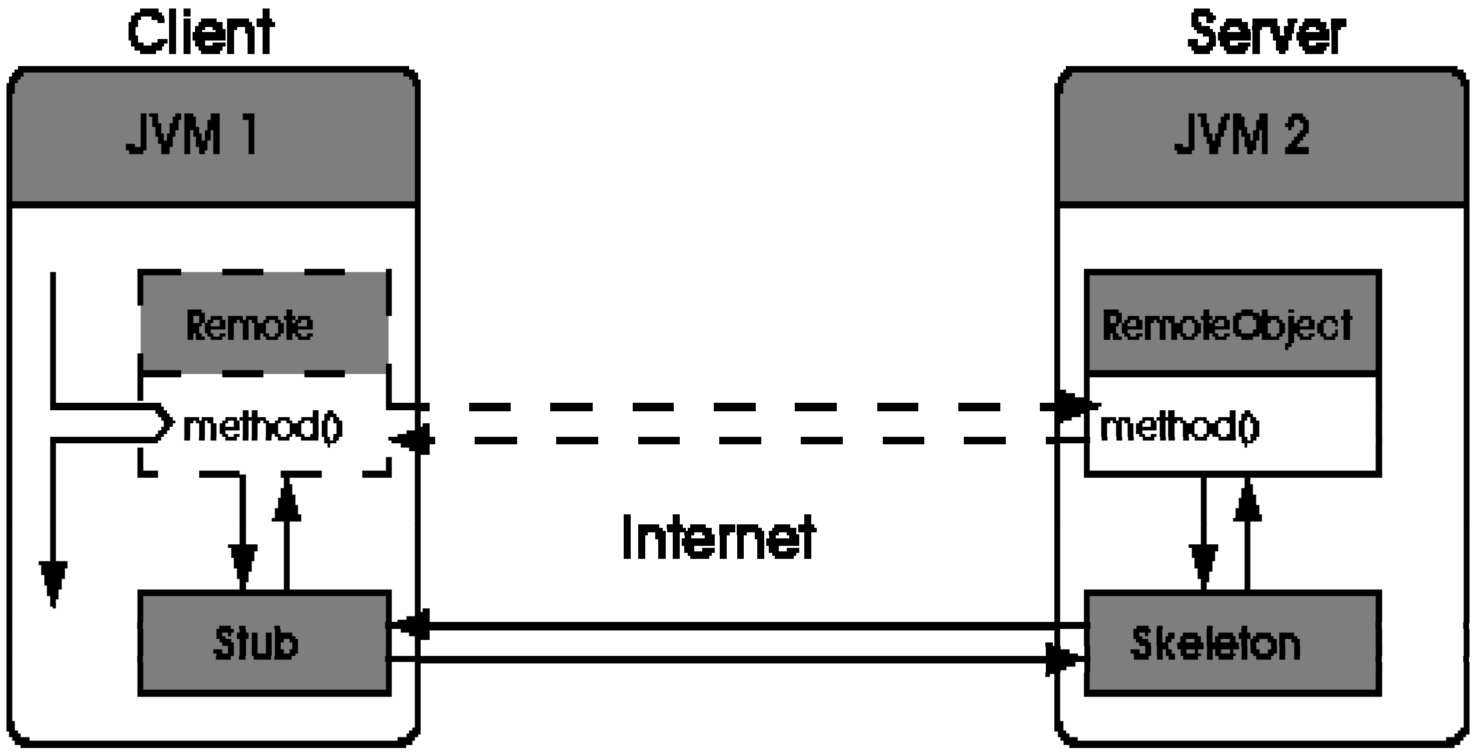
I dettagli relativi alla comunicazione distribuita scompaiono. Tutti metodi sono invocati seguendo i dettami delle invocazioni classiche.

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

***<u>Stub</u>* e *<u>Skeleton</u>* devono essere creati per interfacciare i due oggetti. Sono creati automaticamente (rmic su jdk).**

Lo stub è una classe che traduce ogni invocazione in una comunicazione destinata all'altro oggetto. Lo skeleton è la classe corrispondente che risiede sul server remoto e che accetta le comunicazioni e le ritraduce in invocazioni.

Remote Method Invocation
(schema di invocazione remota)



Remote Method Invocation
(registrazione sul name server)

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

Il cliente contatterà il name server per avere un riferimento all'oggetto remoto. 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 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
(oggetti remoti)

Un *__oggetto remoto__* è un oggetto predisposto per accettare richieste da oggetti in esecuzione su altre java virtual machine.

- **Interfaccia dell'oggetto remoto.**

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

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

```
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 degli oggetti remoti**

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.

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

public class MyRemoteImpl          extends UnicastRemoteObject
                                   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)

```
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);
    }
}
```

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);
    }
}
```

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.

<u>Nota</u> 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 cliente 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.

```
System.setSecurityManager(new RMISecurityManager());
RemoteService remoteService = new RemoteService();
Naming.bind("serviceName", remoteService);
```

Remote Method Invocation
(interfaccia Remote)

L'interfaccia Remote è la superinterfaccia di tutte le interfacce remote. I metodi descritti nelle interfacce remote sono gli unici che possono essere invocati tramite RMI.

- **Il passaggio dei parametri è per valore**
- **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)

La classe Naming fornisce gli strumenti per manipolare l'indirizzamento degli oggetti remoti.

- **Indirizzamento**

  Per indirizzare i servizi viene usato uno schema URL–like:

  RMI://hostname:port/service–name

  Tutti gli elementi tranne il nome del servizio sono opzionali.

- **Remote lookup(String address)**

  throws MalformedURLException, RemoteException, NotBoundException

  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)**

  throws MalformedURLException, RemoteException, AlreadyBoundException

  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 saranno oggetti remoti devono estendere la classe <u>RemoteObject</u>. La classe <u>RemoteObject</u> implementa l'interfaccia <u>Serializable</u>, per permettere la serializzazione degli oggetti remoti.

Se un oggetto remoto è parametro di un'invocazione ad un altro oggetto remoto, in realtà viene inviato un riferimento all'oggetto remoto e non l'oggetto stesso. <u>È un errore definire un metodo remoto che riceve l'implementazione di un oggetto remoto</u>.

```
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);
    }
}
```

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**

Riferimenti Bibliografici

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