Remote Method Invocation in Java

Walter Cazzola

Dipartimento di Informatica e Comunicazione
Università degli Studi di Milano

Outline

1. Remote Procedure Call (RPC)
   - Definizione e Nozioni Generali.
   - Aspetti Nascosti
   - Remote Method Invocation (RMI)
2. RMI in Java
   - Costruire Client e Server in Java
   - Dettaglio Java Library per RMI
   - Remote Class Loading
3. Riferimenti Bibliografici

Remote Procedure Call (RPC)
Introduzione Storica.

Il modello client/server si proponeva come l'approccio principale allo sviluppo di applicazioni distribuite.

- Modello semplice ed intuitivo, rispetto agli altri modelli, ma ancora
- troppo orientato alla comunicazione, era necessario conoscere come e con chi si voleva interagire e molti altri dettagli tecnici legati al canale di comunicazione stesso.

Per garantire l'affermazione del modello era necessaria una maggiore astrazione che nascondesse i dettagli tecnici legati alla comunicazione.

Ad inizio anni 80, Birrell e Nelson introdussero le **Remote Procedure Call (RPC)** come meccanismo di astrazione.

Remote Procedure Call (RPC)
Definizione e Nozioni Generali.

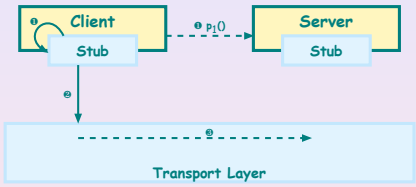
Definizione (Intuitiva).

L'astrazione proposta consiste nell'estendere il meccanismo di attivazione di procedura al caso remoto.

- La richiesta del servizio avviene invocando *trasparentemente* un metodo del server,
- non è necessario inviare un messaggio ne sincronizzarsi sulla ricezione del messaggio di ritorno.

Il concetto di RPC equivale all'occultamento di tutto il codice relativo alla comunicazione di rete.

Remote Procedure Call (RPC)
Definizione e Nozioni Generali.

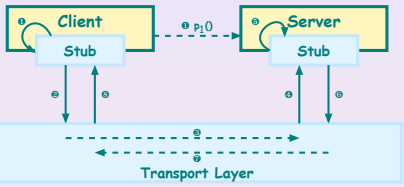


❶ Il client invoca una procedura remota come se fosse locale, in realtà:

- ❶ invoca una procedura wrapper locale al client.
- È cura del wrapper impacchettare (marshaling) in un messaggio gli argomenti per la procedura remota.

❷ Lo stub del client affida il messaggio allo strato di trasporto che ❸ provvede ad inviarlo all'host remoto.

Remote Procedure Call (RPC)
Definizione e Nozioni Generali.



❹ Sul server una procedura dello stub del server è in attesa di messaggi di richiesta, quando arrivano provvede a:

- spacchettare (unmarshalling) il messaggio;
- ❺ eseguire una chiamata locale al corrispondente metodo del server.

Terminata la computazione il risultato è restituito allo stub che lo impacchetterà in un messaggio e lo restituirà al client (passi ❻-❽).

Remote Procedure Call (RPC)
Trasparenza.

Si può invocare una procedura remota come se fosse locale. Alcuni aspetti vanno considerati:

Passaggio di Parametri.

- semplice nel caso di passaggio per valore,
- problemi se per riferimento o che ammettono side-effect.

Client e server hanno spazio di indirizzamento diverso.

Binding.

Si deve poter reperire il server che offre il servizio richiesto, due tecniche:

- un db centralizzato che tenga traccia dei server disponibili in rete e dei loro servizi;
- un server noto per ogni host che tiene traccia dei server e dei servizi su quell'host.

Remote Procedure Call (RPC)
Trasparenza (Cont.).

Eccezioni.

Oltre alle situazioni di errore classiche la chiamata può fallire per motivi legati alla comunicazione.

Semantica della Chiamata.

In locale, l'esito del servizio richiesto è chiaro. Nel distribuito si può non ottenere risposta perché la comunicazione è fallita.

Solo per i servizi idempotenti questo comportamento non è un problema. Serve chiarezza nella semantica della chiamata:

- esattamente una volta;
- al più una volta; o
- almeno una volta.

Marshalling e Unmarshalling.

Le architetture di partenza e arrivo possono essere diverse, i dati scambiati vanno quindi convertiti.

Remote Procedure Call (RPC)
Varianti del Protocollo RPC.

Request (R) Protocol.

Il client effettua la richiesta e procede senza attendere risposta (RPC asincrona, semantica al più una volta).

- usato per procedure che non prevedono valori di ritorno, o
- con meccanismi avanzati per gestire i risultati (es. future).

Request-Reply (RR) Protocol.

Il client effettua la richiesta ed attende il risultato dal server (RPC sincrona).

- la ricezione del valore di ritorno ha anche il ruolo di messaggio di acknowledge (semantica almeno una volta);

Request-Reply-Acknowledge (RRA) Protocol.

Il client richiede un servizio e attende la risposta. Ricevuto il risultato, informa il server dell'avvenuta ricezione.

Remote Method Invocation (RMI)
Nozioni Generali.

RMI permette ad un oggetto di attivare metodi di un altro oggetto (remoto) indipendentemente da dove si trova.

Estende il concetto di RPC all'invocazione dei metodi:

- le osservazioni fatte per RPC continuano a valere;

Però

- il destinatario della richiesta è un oggetto remoto e non (apparentemente) un processo;
- gli argomenti possono essere oggetti locali o a loro volta remoti;
- il metodo invocato può invocare metodi dei suoi argomenti.

I dettagli relativi alla distribuzione sono nascosti e i metodi remoti sono invocati come quelli locali.

Remote Method Invocation (RMI)
Nozioni Generali.

Stub, Skeleton e Metodi Wrapper.

I server remoti devono definire (ed implementare) un'interfaccia (remota) che specifichi i metodi offerti.

Lo **Stub** e lo **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 ed accetta le comunicazioni dal client e le traduce in invocazioni.

Nelle versioni più recenti di Java lo skeleton non è più necessario, le sue funzionalità sono svolte da altre entità.

Remote Method Invocation (RMI)
Registrazione sul Name Server.

Binding.

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 (RMI)
Registrazione sul Name Server (Cont.).

Binding. (Cont.)

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 (RMI)
Invocazione.

Semantica dell'Invocazione.

Ottenuto il riferimento remoto, il cliente può procedere ad invocare i metodi dell'oggetto remoto, come se fossero 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.

Lo stub intercetta le invocazioni a metodi remoti e le gira allo skeleton dell'oggetto target remoto. In modo analogo, i valori calcolati dal server vengono restituiti al client.

RMI in Java
Interfaccia per Oggetti Remoti.

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`.

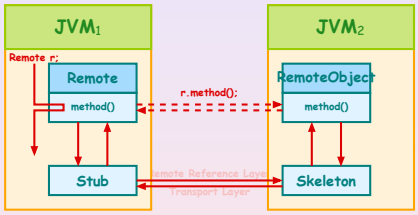
MyRemote.java

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

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


Remote Method Invocation (RMI)
Invocazione.

Semantica dell'Invocazione. (Cont.)



Grazie alla gestione delle eccezioni remote il meccanismo di invocazione remota di Java ha semantica: al più una volta.

RMI in Java
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.

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();
 }
}
```


RMI in Java
Stub & Skeleton.

Generazione di stub & skeleton.

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

```
[22:28]cazzola@ulik:~/SD/RMI> rmic MyRemoteImpl
```

- `rmic`, a partire dal bytecode del server, crea due nuovi file

```
[22:29]cazzola@ulik:~/SD/RMI> ls -l *.class
-rw-r--r-- 1 cazzola collab  408 Jan 12 22:28 MyRemoteImpl.class
-rw-r--r-- 1 cazzola collab 1706 Jan 12 22:28 MyRemoteImpl_Skel.class
-rw-r--r-- 1 cazzola collab 3233 Jan 12 22:28 MyRemoteImpl_Stub.class
```

che contengono il bytecode dello stub e dello skeleton.

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`.

RMI in Java
Registrare e Riferire l'Oggetto Remoto.

Il name server

- per poterne invocare i remoti, i client devono avere un riferimento dell'oggetto remoto;
- fornire i riferimenti è compito del name server (`rmiregistry`);
- un name server deve essere in esecuzione su ogni host coinvolto.

Registrarsi

- gli oggetti remoti si registrano tramite la classe: `java.rmi.Naming` ed adottano uno schema basato su URL per essere identificati.

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

Recupero riferimenti

- I riferimenti sono recuperati tramite la classe: `java.rmi.Naming`.

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


RMI in Java
Esempio Date Server: RMI Client & Server Side.

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

DateServerImpl.java
```
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("DateServer", dateServer);
 }
}
```

DateClient.java
```
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]+"/DateServer");
  Date when = dateServer.getDate();
  System.out.println(when);
 }
}
```


RMI in Java
Interfaccia Remote.

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

I parametri sono passati per valore (si ha side effect sugli oggetti). Le classi dei parametri devono essere serializzabili.

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

Esempio di interfaccia remota:

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


RMI in Java
Classe Naming.

La classe `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`.

RMI in Java
Classe RemoteObject.

Ogni classe, le cui istanze devono essere 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 prevede come parametro o valore di ritorno 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

RMI in Java
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.
  - Oggetto remoto (non replicato) i cui riferimenti sono validi fintanto che è vivo;
  - serializzabile (gli oggetti serializzati sono ancora oggetti remoti);
  - accessibile tramite TCP

Activatable

- Gli oggetti remoti derivati dalla classe `Activatable` sono persistenti nel tempo ed attivabili su richiesta.

RMI in Java
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).

RMI in Java
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.

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

RMI in Java
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.

RMI in Java
Deserializzazione

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

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
}
```


RMI in Java
Serializzazione Oggetti Remoti.

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();
```

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

RMI in Java
Remote Class Loading

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.

RMI in Java
Remote Class Loading

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 non è così.

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.

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.

RMI in Java
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

- **Andrew D. Birrell and Bruce Jay Nelson.**
  Implementing Remote Procedure Calls.
  ACM Transactions on Computer Systems, 2(1):39–59, Feb. 1984.
- **Merlin Hughes, Michael Shoffner, and Derek Hamner.**
  Java Network Programming.
  Manning Publications Co., second edition, May 1999.
- **jGuru.**
  Fundamentals of RMI (Short Course).
  Available on `http://developer.java.sun.com/developer/onlineTraining/rmi/RMI.html`.
- **SUN Microsystems.**
  Java Remote Method Invocation - Distributed Computing for Java.
  White paper, SUN Microsystems, 1998.
  Internet Publication - `http://www.sun.com`.