Java RMI (Remote Method Invocation)

Passaggio parametri per reference o per valore

> **Definizione di Distribuited Objects**
>
> Gli oggetti distribuiti (o "distributed objects" in inglese) sono oggetti di programmazione che possono essere utilizzati e acceduti in un ambiente distribuito, ovvero su più nodi di una rete. In sostanza, sono oggetti che possono esistere su diversi computer e interagire tra di loro attraverso meccanismi di comunicazione, consentendo agli sviluppatori di creare applicazioni distribuite.

Gli oggetti sono caratterizzati da:

1. **Incapsulamento dei dati**:
   Gli oggetti distribuiti incapsulano i dati, cioè il loro stato interno, che può essere costituito da valori dei campi o variabili. Questo significa che gli oggetti mantengono uno stato interno che può essere modificato solo attraverso i metodi definiti nell'oggetto stesso.
2. **Operazioni sui dati**:
   Gli oggetti distribuiti definiscono operazioni sui dati attraverso i metodi delle classi. Questi metodi consentono di eseguire azioni specifiche sugli oggetti e di manipolare il loro stato interno.
3. **Definizione dell'accesso**:
   Gli oggetti distribuiti definiscono l'accesso ai dati e alle operazioni attraverso i metodi delle interfacce. Le interfacce definiscono il contratto che le classi devono implementare, garantendo coerenza nell'accesso e nelle operazioni.

Alcune categorie di Distribuited object sono:

1. **Oggetti a compile-time**:
   Gli oggetti distribuiti definiti a compile-time sono definiti attraverso interfacce e classi. Questo approccio è comune in linguaggi di programmazione come Java e C++, dove le classi e le interfacce vengono definite staticamente durante la fase di compilazione del codice sorgente.
2. **Oggetti a run-time**:
   Gli oggetti distribuiti accessibili a run-time sono oggetti che possono essere acceduti e manipolati dinamicamente durante l'esecuzione del programma. Questi oggetti possono essere ottenuti attraverso adattatori (ad esempio, wrappers o proxy) che consentono l'accesso a oggetti distribuiti dinamicamente.
3. **Oggetti persistenti e transienti**:
   Gli oggetti persistenti mantengono il loro stato anche attraverso riavvii o interruzioni di sistema, mentre gli oggetti transienti perdono il loro stato quando il programma termina o viene riavviato. Gli oggetti persistenti sono spesso utilizzati per mantenere dati critici o condivisi su un server distribuito.
4. **Riferimento agli oggetti remoti**:
   Questo si riferisce al modo in cui gli oggetti distribuiti possono fare riferimento a oggetti che risiedono su un altro nodo della rete, noto come oggetti remoti. I riferimenti agli oggetti remoti consentono agli oggetti di comunicare e collaborare su una rete distribuita.

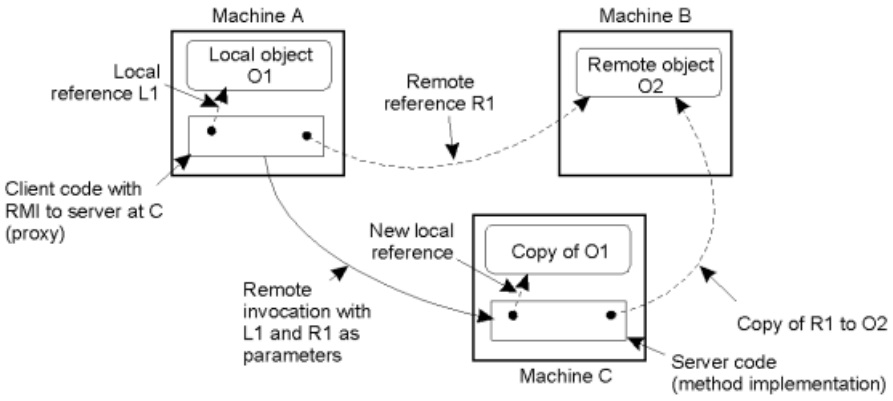
Qual è la differenza tra pointer e riferimento? (C++)

Pointer	Riferimento
Un puntatore è una variabile che contiene un indirizzo di memoria.	Un riferimento (&) è come un alias a una variabile esistente.
Un puntatore può essere inizializzato in qualsiasi momento.	I riferimenti, a differenza dei puntatori, devono essere inizializzati al momento della definizione.

Pointer	Riferimento
Un puntatore può puntare a molti oggetti diversi durante la sua vita.	Un riferimento può riferirsi a un solo oggetto durante la sua vita. In altre parole, un riferimento non può "referenziare" un altro oggetto in un secondo momento.
È possibile creare un array di puntatori.	Non è possibile creare un array di riferimenti perché ogni riferimento dell'array non può essere inizializzato al momento della creazione.
È necessario utilizzare un operatore esplicito (operatore `*`) per de-referenziare un puntatore.	Non è necessario alcun operatore per de-referenziare un riferimento.
Un puntatore non deve necessariamente fare riferimento a un oggetto.	Un riferimento valido deve riferirsi a un oggetto.
Un puntatore, anche un puntatore const, può avere un valore NULL.	Il riferimento non può essere NULL.

Un puntatore distribuito non può esistere (non avrebbe senso).
Un reference distribuito potrebbe essere:

- Indirizzo (IP) della macchina
- Indirizzo (porta) del server (processo)
- Identificatore (ID) dell'oggetto



Middleware RMI

RMI, o Remote Method Invocation, è un middleware che consente di estendere l'approccio orientato agli oggetti al contesto distribuito, consentendo la comunicazione e l'interazione tra oggetti su macchine virtuali distinte.

7.1 Caratteristiche

- **Supporto all'ereditarietà**: RMI supporta l'ereditarietà, consentendo agli oggetti remoti di ereditare da classi di base e implementare interfacce Java.
- **Invocazione di metodi tra oggetti su macchine virtuali distinte**: RMI permette di invocare metodi su oggetti remoti, che possono risiedere su macchine virtuali diverse all'interno di una rete.
- **Interfacce Java**: Le interfacce utilizzate in RMI sono definite in Java stesso, non in un linguaggio di definizione delle interfacce (IDL) generico.
- **Passaggio di oggetti Java**: RMI consente di passare e ritornare oggetti Java tra client e server durante le chiamate remote.
  Quindi, JavaRMI consente la gestione dei parametri per valore e per reference.
- **Caricamento dinamico delle classi**: Le classi utilizzate in RMI possono essere caricate dinamicamente durante l'esecuzione del programma.
- **Basato sulla portabilità del bytecode e sulla macchina virtuale Java**: RMI sfrutta la portabilità del bytecode Java e della macchina virtuale Java per garantire sicurezza ed efficienza, evitando la necessità di traduzioni.

7.1.1 Servizi di Java RMI

Fornitura di servizi: RMI fornisce una serie di servizi, tra cui:

- Caricamento e controllo delle classi tramite un class loader e un security manager.
- Gestione di oggetti remoti, replicati e persistenti.
- Attivazione automatica degli oggetti remoti.
- Supporto per il multi-threading.
- Garbage collection degli oggetti remoti utilizzando un meccanismo di conteggio dei riferimenti.

7.1.1 Ambiente Java

7.1.1.1 Invocazioni

Invocazioni statiche

Le invocazioni statiche in Java RMI implicano che l'interfaccia del metodo remoto è nota al momento della compilazione. Questo significa che i metodi che possono essere invocati sono predefiniti e la loro firma (nome, parametri, tipo di ritorno) è conosciuta.

Invocazioni dinamiche

Le invocazioni dinamiche consentono una maggiore flessibilità, permettendo di invocare metodi senza conoscere la loro firma esatta al momento della compilazione. L'invocazione include informazioni logiche sull'identità dell'oggetto e del metodo. Questo può essere realizzato utilizzando il metodo `java.lang.reflect.Method.invoke`, che permette di invocare un metodo specificato tramite riflessione.

La sintassi generica sarebbe:

```
invoke(obj, method, input_parameters, output_parameters)
```

Qui, `obj` rappresenta l'oggetto su cui invocare il metodo, `method` è un riferimento al metodo da invocare, e `input_parameters` e `output_parameters` sono rispettivamente i parametri di input e output per il metodo.

7.1.1.2 Trasferimento dei parametri

Java RMI è simile a RPC (Remote Procedure Call) in quanto entrambi gestiscono i parametri per valore. Questo significa che i dati passati ai metodi remoti sono copiati e trasmessi al server remoto, dove vengono utilizzati nel metodo invocato.
Tuttavia, A differenza di RPC tradizionali, Java RMI consente anche il passaggio di parametri per riferimento. Questo significa che invece di copiare l'intero oggetto, viene trasmesso un riferimento all'oggetto remoto, permettendo al metodo remoto di operare direttamente su di esso.

```
java.rmi.server.UnicastREmoteObject
```

È una reference class che serve per il trasferimento remoto per reference.
Essa implementa le interfacce `Remote` e `Serializable`.

7.1.1.3 Object serialization

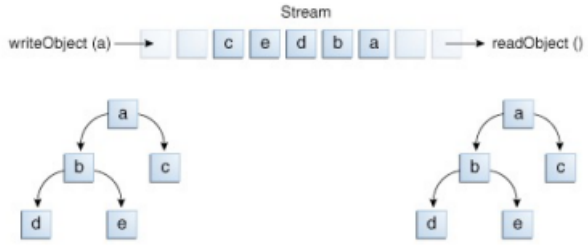
Rappresentazione di uno stato di un oggetto come stream di byte.

Perchè è essenziale memorizzare e ricostruire lo stato degli oggetti?

- Per trasferire oggetti via rete;
- Per definire oggetti persistenti.

Meccanismo di loading dinamico di Java
Il meccanismo di loading dinamico di Java permette di trasferire solo le informazioni essenziali sullo stato di un oggetto, mentre la descrizione della classe può essere caricata separatamente. Questo approccio **ottimizza il processo di trasferimento dei dati**, riducendo il carico di rete e migliorando l'efficienza.

Serializzazione in Java



La serializzazione in Java utilizza il metodo `writeObject(Object obj)` della classe `ObjectOutputStream`. Questo metodo percorre tutti i riferimenti contenuti nell'oggetto `obj` per costruire una rappresentazione completa del grafo degli oggetti.
In altre parole, quando un oggetto viene serializzato, non solo vengono scritti i dati dell'oggetto stesso, ma anche tutti gli oggetti a cui fa riferimento, creando una copia completa e fedele dell'intero grafo di oggetti.

Quindi, ciò che avviene è:

1. **Informazioni essenziali sullo stato**: durante il caricamento dinamico, solo lo stato minimo necessario dell'oggetto viene trasferito. Questo stato include i dati che definiscono l'attuale condizione dell'oggetto.
2. **Descrizione della classe**: struttura della classe dell'oggetto, ovvero il suo bytecode, può essere caricata separatamente dal codice sorgente o da altri componenti necessari per la definizione della classe.
3. `writeObject(Object obj` : metodo per la serializzazione e assicura che tutto il grafo di oggetti riferiti dall'oggetto iniziale venga correttamente serializzato.
   Ogni riferimento a un altro oggetto all'interno dell'oggetto principale viene attraversato e incluso nella rappresentazione serializzata.

Attenzione: i tipi base sono serializzabili in modo nativo.

Come si crea una classe serializzabile?

- Implementando l'interfaccia *Serializable*
- Ridefinendo i metodi
  `private void writeObject(java.io.ObjectOutputStream out) throws IOException;`
  `private void readObject(java.io.ObjectInputStream in) throws IOException, ClassNotFoundException;`

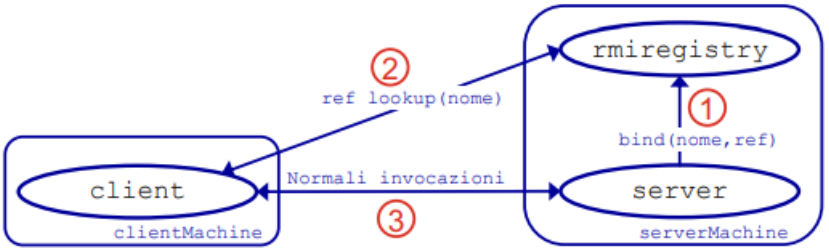
7.1.1.4 Object identification

Come fa eventualmente una macchina a ricavare il riferimento remoto ad un oggetto remoto?

Identificazione degli oggetti
Utilizza nomi assegnati dall'utente e un directory (o naming) service per convertirli in reference operativi. I directory service devono essere disponibile ad un host e porta noti (well-known address).

rmiregistry
RMI definisce un `rmiregistry`, che sta su ogni macchina che ospita oggetti remoti ed è convenzionalmente è alla

porta 1099.
RMI attiva un servizio di ascolto per quel servizio.



L'accesso diretto a `rmiregistry` è permesso dalla classe Naming.

Classe Naming

È una classe che da accesso diretto alle funzionalità di `rmiregistry`.

Caratteristiche

- Metodi static
- I parametri sono stringhe in formato URL riferiti al registry e all'oggetto remoto considerato. `//<host_name>[:<name_service_port>]/]<service_name>`
- Default host: `localhost`
- Default port: `1099`

`public static Remote lookup(String name) throws NotBoundException, MalformedURLException, RemoteException` Restituisce un riferimento, uno stub, all'oggetto associato al nome specificato.

`public static void bind(String name, Remote obj) throws AlreadyBoundException, MalformedURLException, RemoteException`
Collega (bind) il nome specificato all'oggetto remoto.

`public static void rebind(String name, Remote obj) throws RemoteException, MalformedURLException`
Collega (bind) il nome specificato all'oggetto remoto, cancellando i collegamenti esistenti.

`public static String[] list(String name) throws RemoteException, MalformedURLException`
Restituisce i nomi (in formato URL) degli oggetti del registry.

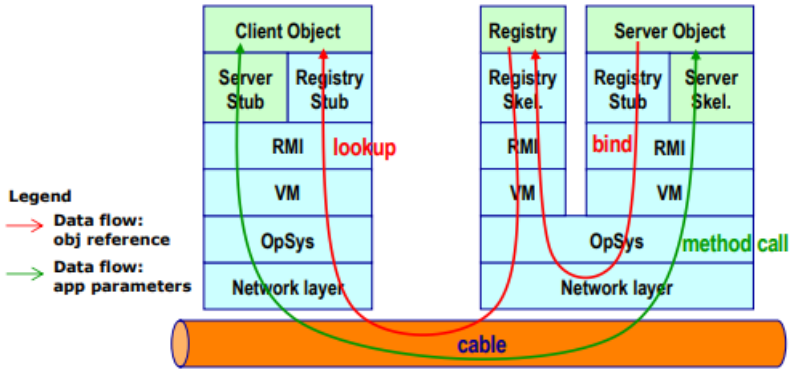
`public static void unbind(String name) throws RemoteException, NotBoundException, MalformedURLException`
Distrugge il collegamento (bind) al nome specificato.

Parametri:

- `name` - a name in URL format (without the scheme component)
- `obj` - a reference for the remote object (usually a stub)

7.1.2 Architettura di RMI

Nell'architettura di RMI, come avviene la pubblicazione e accesso agli oggetti remoti?



1. **Pubblicazione dell'Oggetto Remoto (Server Side)**
   Registrazione nel Registry: Il server pubblica il riferimento e il nome dell'oggetto remoto nel `Registry` utilizzando il metodo `bind`. Questo rende l'oggetto disponibile per i client.

```
Registry registry = LocateRegistry.createRegistry(1099);
registry.bind("NomeOggettoRemoto", oggettoRemoto);
```

2. **Ottenimento del Riferimento all'Oggetto Remoto (Client Side)**
   Lookup: Il client ottiene il riferimento all'oggetto remoto invocando il metodo `lookup` sul `Registry`. Questo permette al client di accedere all'oggetto remoto tramite il suo nome.

```
Registry registry = LocateRegistry.getRegistry("indirizzoServer");
NomeOggettoRemoto stub =
(NomeOggettoRemoto)registry.lookup("NomeOggettoRemoto");
```

3. **Accesso ai Metodi Remoti (Client Side)**
   Invocazione dei Metodi Remoti: Dopo aver ottenuto il riferimento all'oggetto remoto, il client può invocare i metodi remoti su di esso utilizzando il riferimento e i nomi dei metodi remoti.

```
 String risultato = stub.nomeMetodoRemoto(parametri);
```

7.1.2.1 Comunicazione tra diverse VM (JRE)

In Java RMI (Remote Method Invocation), la comunicazione tra le diverse JVM (Java Virtual Machine) avviene attraverso protocolli specifici. Questi protocolli definiscono come i metodi remoti vengono invocati e come i dati vengono trasmessi tra il client e il server.

Protocolli utilizzati nelle diverse versioni di JDK

1. **JDK 1.1 e 1.2: Java Remote Method Protocol (JRMP)**
   In queste versioni, la comunicazione tra le JVM si basa sul Java Remote Method Protocol (JRMP). JRMP è il protocollo nativo di Java RMI che consente la comunicazione diretta tra le JVM, supportando tutte le funzionalità necessarie per invocare metodi remoti.
   Funzionamento: JRMP gestisce la serializzazione degli oggetti e la trasmissione delle chiamate di metodo tra la JVM client e la JVM server, permettendo un'invocazione trasparente dei metodi remoti.
2. **Da JDK 1.3: RMI-IIOP (Internet Inter-ORB Protocol)**
   A partire da JDK 1.3, Java RMI può utilizzare anche RMI-IIOP (Internet Inter-ORB Protocol) per la comunicazione tra le JVM. Questo protocollo consente a Java RMI di interoperare con altre tecnologie CORBA (Common Object Request Broker Architecture).
   Funzionamento: RMI-IIOP permette di invocare metodi remoti non solo tra JVM, ma anche tra applicazioni Java e altri sistemi basati su CORBA. Questo rende Java RMI più versatile, facilitando l'integrazione con un'ampia gamma di sistemi distribuiti eterogenei.

7.1.2.2 Problemi dei sistemi distribuiti

Come Java/RMI affronta le questioni principali dei sistemi distribuiti?

1. **Identificazione della controparte (Naming)**
   Per identificare la controparte in un sistema distribuito, Java RMI utilizza una combinazione di nome dell'host e nome assegnato all'oggetto remoto. Questa combinazione garantisce che ogni oggetto remoto possa essere univocamente identificato e accessibile da altre JVM nel network distribuito.
   Esempio: `rmi://hostname/objectName` permette di individuare e accedere a un oggetto remoto specifico.
2. **Accesso alla controparte** (*Access point*)
   RMI utilizza un registro speciale, chiamato RMI registry, che mappa una coppia `<host, object name>` a un riferimento all'oggetto remoto. Il registro RMI agisce come un punto di accesso centralizzato che consente ai client di trovare i riferimenti agli oggetti remoti.
   - **Esempio**: Un server pubblica l'oggetto remoto nel registro RMI usando `bind`, e il client recupera il riferimento usando `lookup`.
3. **Formato dei dati**
   Java RMI gestisce la comunicazione dei dati tramite la serializzazione degli oggetti Java (sia locali che remoti). La serializzazione converte un oggetto Java in un formato di byte che può essere trasmesso attraverso la rete e ricostruito all'altra estremità. Questo processo consente di trasmettere sia i dati che lo stato degli oggetti complessi, mantenendo l'integrità delle informazioni.
   Esempio: Quando un metodo remoto viene invocato, gli oggetti passati come parametri vengono serializzati, trasmessi al server remoto e deserializzati.
4. **Semantica dei dati**
   Java RMI si basa sui tipi Java, che includono classi e metodi definiti nel codice Java. Questo garantisce che la semantica dei dati e delle operazioni sia coerente su entrambi i lati della comunicazione. Usando le classi e i metodi Java, RMI mantiene una semantica di tipo forte, assicurando che i metodi remoti rispettino le stesse firme e comportamenti definiti nel lato server.
   Esempio: Un'interfaccia remota definisce i metodi che possono essere invocati in remoto, e l'implementazione di questa interfaccia viene distribuita ai client.

Livello di trasparenza
In generale, Java RMI offre un alto livello di trasparenza di accesso e una moderata trasparenza di locazione, facilitando lo sviluppo di applicazioni distribuite come se fossero locali. Tuttavia, ci sono ancora aree in cui il programmatore deve intervenire attivamente.

7.1.2.3 Creazione di un oggetto remoto

Java definisce un'interfaccia per implementare oggetti remoti (server): `interface java.rmi.Remote`.

Per creare una classe remota (server):

1. Definire l'interfaccia della classe remota
   Si estende l'interfaccia Remote
   Il metodo call riceve il messaggio che verrà rimbalzato.
   Nota: ogni metodo deve lanciare `java.rmi.RemoteException`

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

2. Implementare la nuova interfaccia
   Si estende la classe `UnicastRemoteObject`.

```
class java.rmi.server.RemoteObject (implements java.rmi.Remote, java.io.Serializable)

class java.rmi.server.RemoteServer

class java.rmi.activation.Activatable

class java.rmi.server.UnicastRemoteObject
```

3. Implementare un server che crei e registri l'oggetto al Registry

Il server crea, tramite un programma, un oggetto locale che fornisce il servizio desiderato e registra questo oggetto nel registro RMI (RMI registry) sullo stesso host, associandolo a un nome pubblico.

```
MyRemoteObject myObj = new MyRemoteObject();
Registry localRegistry = LocateRegistry().getRegistry();
localRegistry.bind("publicName", myObj);
```