

JAVA RMI Remote Method invocation

1. Interface:

- Contengono solo le firme dei metodi dei servizi a cui fanno riferimento
- Nel caso RMI le classi contengono "extends Remote"
- I metodi richiedono l'osservazione delle eccezioni con "throws RemoteException"

Lato server:

2. ServiceServer:

- Contiene il main del server
- Si occupa di:
 - Creare l'oggetto Registry e farne la LocateRegistry.getRegistry()
 - Istruire un oggetto ServiceImplementation e farne il Bind

3. ServiceImplementation:

- Contiene l'implementazione dei metodi definiti per la sua interfaccia
- Dichiarazione:
 - public Class ServerImpl extends UnicastRemoteObject implements IService
- Importante per il riconoscimento che contenga un parametro: serialVersionUID
- Come ottenerlo:
 - Raggiungere la cartella bin dal terminale
 - Scrivere: serial monospaces.monofileImpl
 - inserire il valore risultante come: private static final long serialVersionUID = ...

Lato Client:

4. ClientThread:

- Si occupa della definizione del comportamento del thread tramite metodo void run()

5. Client:

- È il main client, contiene le fun per l'avvio dei thread e per le seguenti join.

* Tale processo va seguito ogni volta che in un file vogliamo usare dei metodi di un oggetto che implementa la comunicazione tramite RMI!

* Attenzione a servizi già esistenti e che hanno già avuto il loro bind(); in tal caso (Spesso JMS-RMI): IService service = (IService) rmiregistry.lookup("service");

Cose Utili degli appunti al Middleware:

+ Remote procedure Call:

- Comunicazione Sincrona e bloccante tra processi
- **Stub** (Sia su lato client che server) sono le interfacce per la comunicazione
 - ↳ Marshalling ed Unmarshalling
 - Come avviene la conversione dei dati trasmessi?
 - Opzione 1: Sender e Receiver si accordano sul formato
 - Opzione 2: Il sender invia il dato specificandone il formato
 - Qui il numero di punto non viene scelto; si viene a conoscere solo a server attivo.
 - ↳ **Binding Dinamico**
 - **Dispatcher**: Si occupa dell'invio dei messaggi dello Stub Client, all'opportuno Stub Server

+ Message Oriented Middleware:

- Astrazione di coda di messaggi tra processi interoperanti (mailbox)
- Comunicazione peer-to-peer asincrona
- Adatto anche un tipo di comunicazione **publish-subscribe**

+ Modello Spazio delle Tuple:

- Astrazione di uno spazio di memoria associativo dove prelevare e depositare strutture dati (le tuple)
- Richiama le Service Oriented Architecture.
- Necessita di **pattern tuple**

+ Modello ad oggetti distribuiti:

- Estensione in ambiente distribuito della programmazione ad oggetti (DOM = OOP + CS)
- Esempio: **RMI**
 - Invocazione Remota analoga ad RPC (come meccanismo)
 - ↳ Presenta oggetti di sviluppo simili (come gli Stub)
 - **Object Request Brokers** → Bus per la comunicazione (noi li vediamo come interface API)
- **RMI**:
 - Ambiente di elaborazione distribuita omogeneo
 - Server: Crea e registra i servizi nel rmi Registry
 - Client: Utilizza dei servizi registrati

+ Modello a Componenti:

- I componenti sono moduli software autonomamente eseguibili ed omogenei soggetti a composizione. Sono utilizzati per descrivere servizi di entità del mondo reale.
- Possibile associazione (deployment) servizi standard ai componenti; distribuiti tramite container
- **Container**: Ambiente di esecuzione dei componenti (ne filtra anche le richieste in arrivo).
- **JEE**: tecnologie per lo sviluppo di applicazioni Enterprise a componenti

→ Middleware + tale tecnologia = **Application Server**

java RMI

+ RMI:

- Gestione oggetti distribuiti
- Cliente invia messaggio al server per invocare un metodo sull'oggetto servente
- A livello architettonico Client e Server condividono il livello di trasporto
- Le interfacce dei servizi estendono `java.RMI.Remote`
 - L'implementazione è per:
 - Ereditarietà: extends `UnicastRemoteObject` implements ...
 - Delega: implements ...
- Client necessita del riferimento remoto all'oggetto servente
- Si utilizza RMI registry per garantire la trasparenza rispetto alla locazione

Come funziona RMI:

+ Binding:

- Main Server usa il RMI registry per ottenere una locazione su registro ed affidare un nome simbolico all'oggetto Servente

+ Invocazione Remota:

- Client inizializza un oggetto RMI registry ed effettua una lookup per trovare l'oggetto servente

+ Avvio RMI:

- È un servizio fornito dal SO, ergo, avendo raggiunto /bin nel terminale, va prima avviato con il comando `rmiregistry`.

Stub e Skeleton:

+ **Stub:** Creati dinamicamente o automaticamente (però obsoleti)

+ **Skeleton:** Sostituiti da Dispatcher lato server per effettuare le upcalls

Serializzazione:

+ **In/Out:** gli oggetti passati come parametri o come risposta all'invocazione dei metodi remoti sono delle implementazioni dell'interfaccia `Serializable`.

+ **Oggetti:** gli oggetti necessitano di una serializzazione, sono visti come classi che implementano l'interfaccia `Serializable` e che hanno un `SerialVersionUID`

- Viene passata anche la codebase, una URL che punta al bytecode dell'oggetto.
- Utilizzare l'utility `serialver` nomefile per trovare il numero seriale della classe
- Usata dal server per la deserializzazione.

Cenni implementativi:

• Il server avvia un thread per gestire ogni invocazione remota

• L'oggetto remoto è accedito in concorrenza

• RMI sfrutta le socket o socket op stream da noi usata

+ **Callback:** Client crea un callback object remoto da passare al server. Il server lo registra e quando si verifica l'evento di interesse sarà esso ad invocare il callback object.

Ammozioni dagli esercizi:

RMI + Pattern Observer:

- Generator (Client):

- + Main: 1. Lookup del dispatcher remoto
2. Start e Join dei thread

- + Generator thread: 1. Costruttore a cui passare i Dispatcher
2. Run()

- + Reading: Oggetto serializable che viene passato con i metodi remoti

- Interface:

- + iDispatcher

- + iObserver

- Observer:

- + ObserverImpl: 1. Costruttore (idispatcher, parametri)

- 2. Metodo NotifyReading() ← Metodo Remoto

- + Observer: Utilizzo metodi Remoti

- Dispatcher:

- + DispatcherImpl: 1. Costruzione classe

- 2. Implementazione metodi

- + Main: 1. Rebind del servizio

Utilizzare un Servizio Remoto:

1. LocateRegistry.getRegistry()
2. IService service = (IService) rmi.lookup("service")
3. Utilizzo il servizio

} Si fa nel main!

Sottoscrivere un Servizio:

1. IService Service = new ServiceImpl()
2. LocateRegistry.getRegistry()
3. rmi.rebind("service", service)