Java RMI

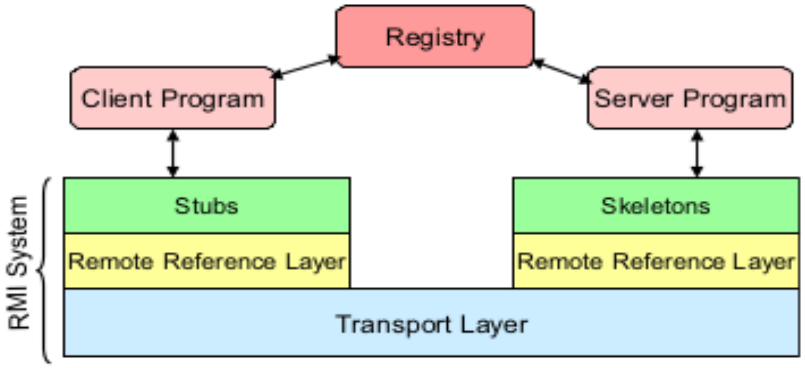
Java rende disponibile un insieme di politiche e meccanismi che permettono ad un'applicazione in esecuzione su una macchina di invocare i metodi di un'altra applicazione Java in esecuzione su una macchina remota. Localmente viene creato solo il riferimento ad un oggetto remoto, che è invece effettivamente attivo su un nodo remoto. Un programma cliente invoca i metodi attraverso questo riferimento locale. Questo è reso possibile tramite l'utilizzo di due proxy, lo stub locale al client e lo skeleton presente sul server. Il client invoca i metodi sullo stub, lo stub invia le invocazioni dei metodi allo skeleton che provvede ad eseguirli sul server.

Il "registry" è un servizio che consente al server di pubblicare un servizio e al client di recuperarne lo stub.

Quindi un'applicazione RMI è formata da server, che crea alcuni oggetti remoti, ne rende accessibili i riferimenti e attende connessioni dai client, e client, che ricevono riferimenti di oggetti remoti e ne utilizzano i metodi, realizzando un'applicazione ad oggetti distribuita.

Ovviamente i client non conoscono l'implementazione degli oggetti remoti, ma solo un'interfaccia con la definizione dei metodi remotamente accessibili.

Questa architettura consente di mascherare alle applicazioni e dettagli di rete sottostanti.



Lo strato skeleton/stub è l'interfaccia tra lo strato di applicazione e il resto del sistema RMI.
Uno stub per un oggetto remoto è il proxy lato client per l'oggetto remoto e implementa tutte le interfacce supportate dall'implementazione oggetto remoto; è responsabile di:

- avviare l'invocazione del metodo sull'oggetto remoto;
- organizzare l'invio dei parametri al metodo;
- gestire la ricezione del valore di ritorno o eventuali eccezioni provocate dall'esecuzione del metodo.

Uno skeleton per un oggetto remoto è un'entità lato server che riceve le invocazioni remote e le 'gira' all'oggetto effettivo; è responsabile di:

- ricevere gli argomenti dallo stub;
- attivare la chiamata sull'oggetto;
- predisporre e inviare allo stub il valore di ritorno.

Le classi stub e skeleton sono generate utilizzando il compilatore rmic, vengono determinate in fase di esecuzione e sono caricate dinamicamente a seconda delle necessità.
In questa architettura il Remote Reference Layer fornisce il supporto alle chiamate inoltrate dallo stub e definisce e supporta la semantica dell'invocazione e della comunicazione, mentre il Transport Layer localizza il server RMI relativo all'oggetto remoto richiesto, gestisce le connessioni (TCP/IP,

timeout) e le trasmissioni (sequenziali, serializzate), usando un protocollo proprietario.
In generale, il livello di trasporto RMI è responsabile di aprire e gestire le connessioni verso il nodo remoto e ricevere e gestire chiamate in arrivo.
Il trasporto per il sistema RMI di Java consiste in quattro astrazioni:

1. *endpoint:* rappresenta uno spazio di indirizzi di una JVM, un'istanza di trasporto si ottiene per un endpoint;
2. *channel:* è l'astrazione per un canale di comunicazione tra due endpoint, serve a gestire le connessioni tra gli spazi degli indirizzi di due JVM;
3. *connection*: è l'astrazione per il trasferimento dei dati;
4. *transport*: gestisce tutti i canali, accetta le chiamate in arrivo, attiva un *connection object* per la chiamata.

Questo strato normalmente si appoggia al TCP, ma potrebbe anche essere implementato anche su UDP.
Abbiamo dunque un modello a oggetti distribuito. Un **oggetto remoto** è un oggetto i cui metodi sono invocabili da un'altra JVM ed è descritto tramite un interfaccia remota, che dichiara i metodi remotamente accessibili. Si deve tener conto che le chiamate remote, a differenza di quelle locali, sono asincrone e possono fallire per problemi legati alla connessione di rete.

L'interfaccia remota deve estendere java.rmi.Remote e propagare java.rmi.RemoteException, l'oggetto remoto oltre a implementare l'interfaccia definita, deve estendere java.rmi.UnicastRemoteObject.

Per sviluppare un'applicazione distribuita usando RMI si deve:

- definire interfacce e implementazioni dei componenti utilizzabili in remoto ;
- compilare le classi (con javac) e generare stub e skeleton (con rmic) delle classi utilizzabili in remoto.
- pubblicare il servizio :
  - attivare il registry
  - registrare il servizio (il server deve fare una bind sul registry)
- il client deve ottenere il riferimento all'oggetto remoto tramite il name service, facendo una lookup sul registry .

A questo punto l'interazione tra il cliente e il server può procedere .

Passaggio dei parametri nei metodi remoti.

Tipo	Metodo locale	Metodo remoto
Tipi primitivi	Per valore	Per valore
Oggetti	Per riferimento	Per valore deep copy
Oggetti remoti esportati	Per riferimento	Per riferimento remoto

Quando un metodo remoto prevede di passare come parametri degli oggetti o ricevere un oggetto come valore di ritorno, è ovvio che non avrebbe senso lo scambio di riferimenti, visto che client e sever non condividono la memoria.
Client e server devono in questo caso scambiarsi delle copie degli oggetti, ricorrendo (in modo trasparente) alla serializzazione.

La serializzazione è un'attività che permette di salvare un oggetto o un insieme di oggetti in uno stream di byte che successivamente può essere salvato in maniera persistente ad esempio in un file

oppure inviato sulla rete. Questo processo coinvolge solo le variabili d'esemplare, escluse quelle costanti e quelle dichiarate con l'attributo transient.
La deserializzazione è il processo inverso, che permette, cioè, di ricostruire l'oggetto o il grafo di oggetti dallo stream di byte. Per farlo è necessario disporre della classe.
Una classe serializzabile deve implementare l'interfaccia java.io.Serializable e tutte le sua variabili d'esemplare devono essere a loro volta serializzabili.
Inoltre è opportuno, anche se non indispensabile, inserire nella classe un numero di versione, ad esempio:

```
private static final long serialVersionUID = 1;
```

che è usato durante la deserializzazione per verificare che la versione della classe caricata da che trasmette e da chi riceve siano la stessa e che quindi gli oggetti sono compatibili relativamente alla serializzazione.

Concorrenza

RMI permette a più client di utilizzare l'oggetto remoto, gestendo in modo automatico la concorrenza delle applicazioni, l'implementazione dei metodi remoti quindi deve tener conto della possibile concorrenza degli accessi, eventualmente utilizzando metodi sincronizzati o blocchi.