Lezione n.9
LPR-A-09
RMI CallBacks

15/12/2009

Vincenzo Gervasi

Sommario

- RMI: il meccanismo delle callbacks

- Thread Miscellanea:
  - Thread Pool: politiche di saturazione
  - Blocchi Sincronizzati

RMI: PASSAGGIO DI PARAMETRI

- Nell'invocazione di un metodo con RMI, i parametri vengono passati nel seguente modo:

  - parametri di tipo primitivo vengono passati per valore
  - parametri di tipo riferimento vengono serializzati e il server ne crea una copia
    - Qundi una modifica fatta dal server non è visibile dal client
  - parametri di tipo "remoto" vengono passati come riferimenti

IL MECCANISMO DELLE CALLBACK: MOTIVAZIONI

Meccanismo RMI prevede:

- comunicazione **unidirezionale** (dal client al server)
- comunicazione **sincrona**, rendez-vouz esteso: il client invoca un metodo remoto e si blocca finchè il metodo non termina

Ma in molte applicazioni

- il client è interessato ad un evento che si verifica sul server e notifica il suo interesse al server (per esempio utilizzando RMI)
- il server **registra** che il client è interessato in quell'evento
- quando l'evento si verifica, **il server notifica** ai clients interessati l'accadimento dell'event

IL MECCANISMO DELLE CALLBACK: MOTIVAZIONI

Esempi di applicazioni:

- Un utente partecipa a un gruppo di discussione (es: Facebook) e vuol essere avvertito quando un nuovo utente entra nel gruppo.
- Lo stato di un gioco multiplayer viene gestito da un server. I giocatori notificano al server le modifiche allo stato del gioco. Ogni giocatore deve essere avvertito quando lo stato del gioco subisce delle modifiche.
- Gestione distribuita di un'asta: un insieme di utenti partecipa ad un'asta distribuita. Ogni volta che un utente fa una nuova offerta, tutti i partecipanti all'asta devono essere avvertiti.

IL MECCANISMO DELLE CALLBACK: MOTIVAZIONI

- Come può essere avvisato il client che un evento si è verificato sul server?

  - **Polling**: il client interroga ripetutamente il server, per sapere se si è verificato l'evento atteso. L'interrogazione può avviene tramite mediante **RMI**
    - Svantaggio: inefficiente, spreco di risorse del sistema

  - **Registrazione** dei clients interessati agli eventi e successiva notifica (asincrona) del verificarsi dell'evento al client da parte del server
    - Problema: quale meccanismo può usare il server per avvisare il client?

RMI: IL MECCANISMO DELLE CALLBACK

- Il **meccanismo delle callback** permette di utilizzare **RMI** sia per l'invocazione client-server (registrazione del client) che per quella server-client (notifica del verificarsi di un evento).
- Come funziona?
  - Il server definisce un'interfaccia remota **ServerInterface** con un **metodo remoto** che serve al client per **registrarsi**
  - Il client definisce un'interfaccia remota **ClientInterface** che definisce un **metodo remoto** usato dal server per **notificare** un evento al client
  - Il client conosce la **ServerInterface** e ottiene il puntatore all'**oggetto remoto** tramite il registry

RMI: IL MECCANISMO DELLE CALLBACK

- Il client invocando il **metodo remoto** per **registrarsi** passa al server un riferimento **RC** a un oggetto che implementa la **ClientInterface**
- Il server memorizza **RC** in una sua struttura dati (ad esempio, un Vector)
- Quando deve notificare, il server utilizza **RC** per invocare il **metodo remoto** di notifica definito dal client.
- In questo modo si rende 'simmetrico' il meccanismo di RMI, ma…
  - il client **non registra** l'oggetto remoto in un rmiregistry, ma passa un riferimento a tale oggetto al server, al momento della registrazione

CHE SIGNIFICA "CALLBACK"?

Da Wikipedia: "In programmazione una callback è una **funzione specializzata** che viene passata come **parametro** a **un'altra funzione** (che invece è **generica**). Questo permette alla funzione generica di compiere un lavoro specifico attraverso la callback."

- **Esempio**: la funzione generica **quicksort** prende come argomento l'array da ordinare e una funzione **callback** per confrontare gli elementi.
- Nel contesto Object Oriented, una **callback** è un'istanza che fornisce un'implementazione di un metodo specificato in un'interfaccia.
- **Esempio nelle API Java**: Uso di **Comparator** nei metodi di **sorting** della classe **Arrays**

CALLBACKS: UN ESEMPIO

Lato Server:

- Interfaccia remota che definisce metodi per: (1) Contattare il server: metodo SayHello( ) (2) **Registrare/cancellare** una **callback:**
- Implementazione dell'interfaccia
- Esportazione e pubblicazione su registry di oggetto remoto

Lato Client:

- Intefaccia remota che definisce metodo remoto (callback)
- Implementazione di interfaccia remota
- Programma principale che:
  - Registra una callback presso il server: sarà usata per notificare al client i contatti stabiliti da clients con il metodo SayHello( ).
  - Effettua un numero casuale di richieste del metodo SayHello( )
  - Cancella la propria registrazione

L'INTERFACCIA REMOTA DEL SERVER

```
import java.rmi.*;
    public interface CbServerInt extends Remote {

    /* metodo di notifica */
    public String sayHello(String name) throws RemoteException;

    /* registrazione per il callback */
    public void registerForCallback(CbClientInt callbackClient)
                    throws RemoteException;
    /* cancella registrazione per il callback */
    public void unregisterForCallback(CbClientInt callbackClient)
                    throws RemoteException;
}
```

L'IMPLEMENTAZIONE DEL SERVER

```
import java.rmi.*; import java.rmi.server.*;import java.util.*;
public class CbServerImpl extends UnicastRemoteObject
                                          implements CbServerInt{
    /* lista dei client registrati */
    private List<CbClientInt> clients;

    /* crea un nuovo servente */
    public CbServerImpl( ) throws RemoteException {
        clients = new ArrayList<CbClientInt>( );
    }
    /* continua */
```

L'IMPLEMENTAZIONE DEL SERVER

```
public synchronized void registerForCallback(CbClientInt callbackClient)
                                        throws RemoteException{
    if (!clients.contains(callbackClient)) {
        clients.add(callbackClient);  }
    System.out.println(" New client registered." ); }
/* annulla registrazione per il callback */
public synchronized void unregisterForCallback(CbClientInt callbackClient)
                                        throws RemoteException{
    if (clients.remove(callbackClient)) {
        System.out.println("Client unregistered");
    }else{
        System.out.println("Unable to unregister client.");
    }}
```

L'IMPLEMENTAZIONE DEL SERVER

```
/* metodo di notifica
 * quando viene richiamato, fa il callback a tutti i client registrati */
public String sayHello (String name) throws RemoteException {
      doCallbacks(name);
      return "Hello, " + name + "!";   }
private synchronized void doCallbacks(String name) throws RemoteException{
      System.out.println("Starting callbacks.");
      Iterator<CbClientInt> i = clients.iterator( );
      int numeroClienti = clients.size( );
      while (i.hasNext()) {
            CbClientInt client = i.next();
            client.notifyMe(name); }
      System.out.println("Callbacks complete."); } }
```

L'ATTIVAZIONE DEL SERVER

```
import java.rmi.server.*; import java.rmi.registry.*;
public class CbServer   {
      public static void main(String[  ] args) {
            try { /* registrazione presso il registry */
                  System.out.println("Binding CallbackHello");
                  CbServerImpl server =  new CbServerImpl(    );
                  String name = "CallbackHelloServer";
                  Registry registry = LocateRegistry.getRegistry ("localhost",2048);
                  registry.rebind  (name, server);
                  System.out.println("CallbackHello bound");
            } catch (Exception e) {
                  e.printStackTrace();
                  System.exit(-1); } } }
```

L'INTERFACCIA DEL CLIENT

```
import java.rmi.*;
public interface CbClientInt extends Remote {
    /*  Metodo invocato dal  server per effettuare
        una callback a un client remoto. */
public void notifyMe(String message) throws RemoteException;
}
```

notifyMe(...)

è il metodo esportato dal client e che viene utilizzato dal server per la notifica di un nuovo contatto da parte di un qualsiasi client. Viene notificato il nome del client che ha contattato il server.

L'IMPLEMENTAZIONE DEL CLIENT

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

public class CbClientImpl  extends UnicastRemoteObject
                                        implements CbClientInt {
/* crea un nuovo callback client */
     public CbClientImpl( ) throws RemoteException {
          super( ); }

/* metodo che può essere richiamato dal servente */
     public void notifyMe(String message) throws RemoteException {
          String returnMessage = "Call back received: " + message;
          System.out.println( returnMessage);  } }
```

ATTIVAZIONE DEL CLIENT

```
import java.rmi.*; import java.rmi.server.*; import java.rmi.registry.*;
public class CbClient {
    public static void main(String args[ ]) {
    try {System.out.println("Cerco CallbackHelloServer");
        Registry registry = LocateRegistry.getRegistry("localhost", 2048);
        String name = "CallbackHelloServer";
        /* crea stub di oggetto remoto */
        CbServerInt h =(CbServerInt) registry.lookup(name);
        /* si registra per il callback */
        System.out.println("Registering for callback");
        CbClientImpl callbackObj = new CbClientImpl( );
        h.registerForCallback(callbackObj);
```

ATTIVAZIONE DEL CLIENT

```
        /* accesso al server - fa una serie casuale di 5-15 richieste */
        int n = (int) (Math.random( ) * 10 + 5);
        String nickname=  "mynick";
        for (int i=0; i< n; i++) {
              String message = h.sayHello(nickname);
              System.out.println( message);
              Thread.sleep(1500);  }
        /* cancella la registrazione per il callback */
        System.out.println("Unregistering for callback");
        h.unregisterForCallback(callbackObj);
} catch (Exception e) {
System.err.println("HelloClient exception: " +e.getMessage(  ));
e.printStackTrace(); System.exit(-1); }}}
```

RMI:ECCEZIONI

- Eccezione che viene sollevata se non trova un servizio di registry su quella porta. Esempio:

  HelloClient exception: Connection refused to host: 192.168.2.103; nested exception is: java.net.ConnectException: Connection refused: connect

- Eccezione sollevata se si tenta di registrare più volte lo stesso stub con lo stesso nome nello stesso registry

  Esempio

  CallbackHelloServer exception: java.rmi.AlreadyBoundException: CallbackHelloServer java.rmi.AlreadyBoundException: CallbackHelloServer

ESERCIZIO 1: ELEZIONI CON CALLBACK

La scorsa lezione precedente è stato assegnato un esercizio per la gestione elettronica di una elezione a cui partecipano un numero prefissato di candidati. Si chiedeva di realizzare un server RMI che consentisse al client di votare un candidato e di richiedere il numero di voti ottenuti dai candidati fino ad un certo punto.

Modificare l'esercizio in modo che il server notifichi ogni nuovo voto ricevuto a tutti i clients che hanno votato fino a quel momento. La registrazione dei clients sul server avviene nel momento del voto.

ESERCIZIO 2: GESTIONE FORUM

Si vuole implementare un sistema che implementi un servizio per la gestione di forum in rete. Un forum è caratterizzato da un argomento su cui diversi utenti, iscritti al forum, possono scambiarsi opinioni via rete.

Il sistema deve prevedere un server RMI che fornisca le seguenti funzionalità:

a) apertura di un nuovo forum, di cui è specificato l'argomento (esempio: giardinaggio)

b) registrazione ad un forum, di cui è specificato l'argomento

c) inserimento di un nuovo messaggio indirizzato ad un forum identificato dall'argomento (es: è tempo di piantare le viole, indirizzato al forum giardinaggio); il messaggio deve essere inviato agli utenti iscritti al forum

d) reperimento dell'ultimo messaggio inviato ad un forum di cui è specificato l'argomento.

Quindi il messaggio può essere richiesto esplicitamente dal client oppure può essere notificato ad un client precedentemente registrato.