Lezione n.10
LPR
RMI CallBacks

12/05/2008

Laura Ricci

RMI CALLBACKS

Un client può richiedere servizi ad un servente mediante RMI. Talvolta è utile poter consentire al servente di contattare il client

- il servente notifica degli eventi ai propri client per evitare il polling effettuato dai clients
- il servente accede allo stato della sessione se questo è memorizzato presso il client meccanismo, analogo a quello dei cookie

La soluzione è offerta dal meccanismo delle callback in cui il servente può richiamare il client rendendo la comunicazione bidirezionale

RMI CALLBACKS

- Il client crea un oggetto remoto, oggetto callback *OC*, che implementa un'interfaccia remota che deve essere nota al servente
- Il servente definisce un oggetto remoto *OS*, che implementa una interfaccia remota che deve essere nota al client
- Il client reperisce *OS* mediante il meccanismo di lookup di un registry
- *OS* contiene un metodo che consente al client di registrare il proprio *OC* presso il server
- quando il servente ne ha bisogno, può contattare ***OC***, reperendo un riferimento ad *OC* dalla struttura dati in cui lo ha memorizzato al momento della registrazione

CALLBACKS: UN ESEMPIO

Server:

- Definisce un oggetto remoto che fornisce ai clients metodi per
  - Registrare/cancellare una callback
  - Contattare il server (metodo SayHello)

Client

- Registra una callback presso il server. La callback consente al server di notificare ai clients registrati ogni contatto stabilito dai clients mediante il metodo SayHello
- Effettua un numero casuale di richieste del metodo sayHello
- Cancella la propria registrazione

L'INTERFACCIA DEL CLIENT

```
import java.rmi.*;
public interface CallbackHelloClientInterface 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'INTERFACCIA DEL CLIENT

```
import java.rmi.*; import java.rmi.server.*;
public class CallbackHelloClientImpl
                implements CallbackHelloClientInterface {
                 /* crea un nuovo callback client */
                 public CallbackHelloClientImpl( ) 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); }
                 }
```

IL CODICE DEL CLIENT

```
import java.rmi.*;
public class CallbackHelloClient {
public static void main(String args[ ]) {
      try {
                  vedi lucido successivo.....

        } catch (Exception e) {
           System.err.println("HelloClient exception: " +e.getMessage());
                                    }
          }
       }
```

IL CODICE DEL CLIENT

```
System.out.println("Cerco CallbackHelloServer");
Registry registry = LocateRegistry.getRegistry("localhost", 2048);
String name = "CallbackHelloServer";
CallbackHelloServerInterface h =
        (CallbackHelloServerInterface) registry.lookup(name);
/* si registra per il callback */
System.out.println("Registering for callback");
CallbackHelloClientImpl callbackObj = new CallbackHelloClientImpl( );
CallbackHelloClientInterface stub = (CallbackHelloClientInterface)
                    UnicastRemoteObject.exportObject(callbackObj, 0);
h.registerForCallback(stub);
```

IL CODICE 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);
```

L'INTERFACCIA DEL SERVER

```
import java.rmi.*;
public interface CallbackHelloServerInterface extends Remote {
    /* metodo di notifica */
    public String sayHello(String name) throws RemoteException;
    /* registrazione per il callback */
    public void registerForCallback(CallbackHelloClientInterface callbackClient)
    throws RemoteException;
    /* cancella registrazione per il callback */
    public void unregisterForCallback(CallbackHelloClientInterface callbackClient)
    throws RemoteException;
```

L'IMPLEMENTAZIONE DEL SERVER

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

public class CallbackHelloServerImpl implements CallbackHelloServerInterface
                        /* lista dei client registrati */
                        private List<CallbackHelloClientInterface> clients;
                        /* crea un nuovo servente */
        public CallbackHelloServerImpl( ) throws RemoteException {
                super();
                clients = new ArrayList<CallbackHelloClientInterface>( );
        }
```

L'IMPLEMENTAZIONE DEL SERVER

```
public synchronized void registerForCallback
(CallbackHelloClientInterface 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
( CallbackHelloClientInterface 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 + "!";
                    }
```

IL SERVER:IMPLEMENTAZIONE

```
private synchronized void doCallbacks(String name ) throws
    RemoteException {
      System.out.println("Starting callbacks.");
      Iterator i = clients.iterator( );
      int numeroClienti = clients.size( );
      while (i.hasNext()) {

            CallbackHelloClientInterface client =

                              (CallbackHelloClientInterface) i.next();

            client.notifyMe(name);
      } System.out.println("Callbacks complete.");}}
```

IL CODICE DEL SERVER

```
import java.rmi.server.*; import java.rmi.registry.*;
public class CallbackServer  {
public static void main(String[  ] args) {
try { /* registrazione presso il registry */
      System.out.println("Binding CallbackHello");
      CallbackHelloServerImpl server =  new CallbackHelloServerImpl();
      CallbackHelloServerInterface stub=(CallbackHelloServerInterface)
                  UnicastRemoteObject.exportObject (server, 39000);
       String name = "CallbackHelloServer";
      Registry registry = LocateRegistry.getRegistry ("localhost",2048);
      registry.bind  (name, stub);
      System.out.println("CallbackHello bound");
} catch (Exception e) { System.out.println("Eccezione" +e);   }}}}
```

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 e 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: CALLBACKS

In una 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: 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 possono scambiarsi opinioni via rete. Il sistema deve prevedere un server RMI che fornisca le seguenti funzionalita':

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

b) inserimento di un nuovo messaggio indirizzato ad un forum identificato dall'argomento (es: è tempo di piantare le viole, indirizzato al forum giardinaggio)

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

ESERCIZIO: GESTIONE FORUM

Si devono definire almeno le seguenti classi:

- una interfaccia ForumInterface, che definisce i metodi del server che possono essere invocati in remoto

- una classe Forum che implementa la struttura dati relativa ad un singolo forum. Il numero massimo di messaggi per ogni forum e'definito staticamente ed è uguale per ogni forum.

- una classe Forumimpl che implementa i metodi definiti nella interfaccia ForumInterface. Il numero massimo di forum che possono essere gestiti da Server è determinato staticamente.

ESERCIZIO:GESTIONE FORUM

- Una classe ForumObj che attiva una istanza dell'oggetto remoto Forumimpl

- una classe Client che interagisce con l'utente ed, in base alle richieste dell'utente, invoca i metodi di Server.

- una classe Messaggio che implementa la struttura dati che descrive il messaggio inviato al forum .