

RMI - REMOTE METHOD INVOCATION

Un esempio funzionante

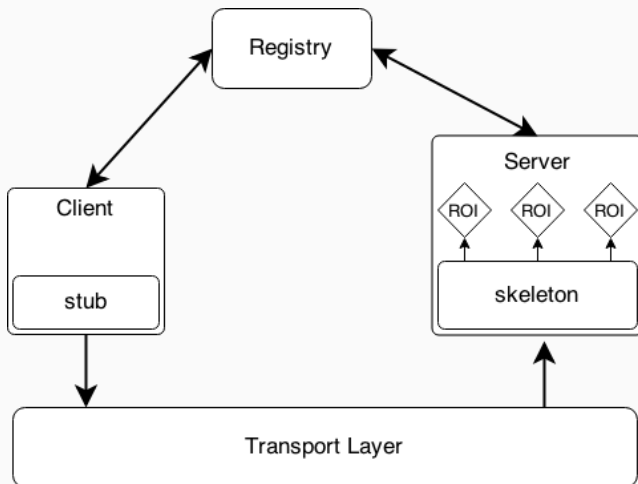
Gabriele Vailati - Tarcisio Zago

Università degli Studi di Milano

INTRODUZIONE A RMI

- Evoluzione Object Oriented del paradigma RPC sviluppata da Sun Microsystems
- Paradigma distribuito che permette ad un'applicazione Java di invocare metodi di oggetti attivi su una JVM differente.
- Adotta il modello client/server.

ARCHITETTURA DI UN'APPLICAZIONE RMI



- Client** il processo che invoca i metodi remoti
- Server** il processo che mette a disposizione un servizio o una risorsa
- Registry** naming service (RMI registry) su cui saranno effettuare le operazioni di bind e lookup
- Stub** rappresentazione locale dell'oggetto remoto, si connette allo skeleton e invia le richieste del client
- Skeleton** riceve le richieste dello stub, le inoltra all'implementazione locale del metodo e restituisce i valori di ritorno

1. Pubblicazione del servizio:
 - esportazione dell'oggetto remoto su una porta (1100)
 - creazione registro su un'altra porta (1099)
 - *bind(NomeServizio, RiferimentoOggetto)* su RMI registry
2. Attende le richieste del client
3. Delega l'elaborazione della richiesta ad un server thread
4. Torna nello stato di attesa (passo 2)

1. Si connette al server
2. *lookup(NomeServizio)* su RMI registry
3. Download stub
4. Invocazione metodo remoto
5. Attesa della risposta

- Naming service attraverso la quale il server registra i servizi che offre (**bind**)
- Il client lo interroga per sapere quali sono i servizi (**lookup**)
- Struttura dati in cui ogni entry è una coppia (*Nomeservizio*, *RiferimentoOggetto*)

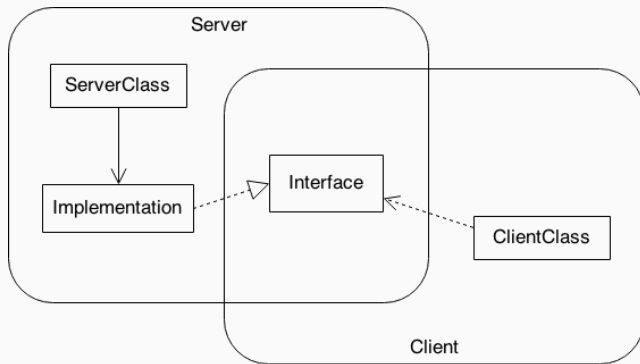
- L'invocazione remota avviene con la stessa sintassi delle invocazioni locali
- Nessun bisogno di un IDL specifico
- Interfacce remote definite con le espressioni native del linguaggio
- Possibilità di passare come argomento tipi di dato primitivi e oggetti locali

- Tipo di dato primitivo
 - *call-by-value*
 - marshalling e unmarshalling dei dati
- Oggetto locale
 - *call-by-reference* non possibile → simulazione
 - serializzazione degli oggetti in uscita
 - trasformazione dell'oggetto in una sequenza di byte
 - deserializzazione degli oggetti in entrata
 - ricostruzione dell'oggetto a partire dallo stream di byte

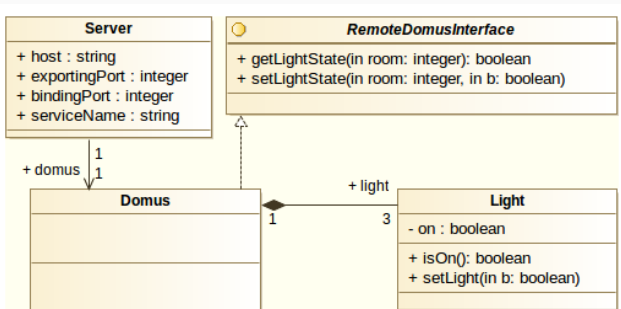
- Le invocazioni remote sono *asincrone*
- Possibilità di fallimento
- Semantica *at-most-once*:
 - Il metodo non è stato eseguito il server invia un messaggio di errore → il client ripete la richiesta
 - altrimenti il metodo è assunto come eseguito (i metodi void non ritornano nessun valore)

ESEMPIO DI IMPLEMENTAZIONE DI UN SERVIZIO

DISTRIBUZIONE DELLE CLASSI



- L'oggetto remoto esposto rappresenta il sistema domotico di un'abitazione
- Per lo scopo di questa presentazione, si mostrerà l'implementazione di due semplici metodi per ottenere e modificare lo stato delle luci



INTERFACCIA REMOTA

```
1 package DomusDemo;
2 import java.rmi.Remote;
3 import java.rmi.RemoteException;
4
5 public interface RemoteDomusInterface extends Remote{
6
7     public boolean getLightState(int room) throws RemoteException;
8     public void setLightState(int room, boolean b) throws RemoteException;
9
10 }
```

METODI ESPOSTI DA DOMUS

```
7   Light[] light;
8
9   public Domus() {
10       light = new Light[3];
11       light[0] = new Light(); //luce prima stanza
12       light[1] = new Light(); //luce seconda stanza
13       light[2] = new Light(); //luce terza stanza
14   }
15
16   @Override
17   public boolean getLightState(int room) throws RemoteException {
18       return light[room].isOn();
19   }
20
21   @Override
22   public void setLightState(int room, boolean b) throws RemoteException {
23       light[room].setLight(b);
24   }
25 }
```


IMPLEMENTAZIONE DI LIGHT

```
1 package DomusDemo;
2
3 public class Light{
4     private boolean on;
5
6     public Light() {
7         on = false;
8     }
9
10    public boolean isOn() {
11        return on;
12    }
13
14    public void setLight(boolean b) {
15        on = b;
16    }
17 }
```

SERVER RMI

Il principale compito del server RMI è quello di pubblicare il servizio sul **naming service**.

Per farlo dovrà predisporre la JVM ad accettare connessioni in entrata su una porta specifica.

Per questo scopo sono usate le classi

- **Registry** l'interfaccia degli oggetti remoti
- **LocateRegistry** fornisce i metodi per ottenere un riferimento all'oggetto remoto
- **UnicastRemoteObject** permette di esportare un oggetto remoto e di ottenere uno stub su cui effettuare le chiamate

SERVER RMI (2)

Per prima cosa, il server crea l'oggetto di cui esporrà i metodi e definisce alcuni attributi che serviranno alla pubblicazione del servizio.

```
10 public class Server {  
11     Domus domus;  
12     static String host = "<hostname>";  
13     static int exportingPort = 1100;  
14     static int bindingPort = 1099;  
15     static String serviceName = "Domus";  
16  
17     public Server() {  
18         super();  
19         domus = new Domus();  
20     }
```

SERVER RMI (3)

```
22 public static void main(String[] args) {
23     Server server = new Server();
24
25     System.setProperty("java.rmi.server.hostname", host);
26     try {
27         RemoteDomusInterface remoteObjectStub =
28             (RemoteDomusInterface) UnicastRemoteObject
29                 .exportObject(server.domus, exportingPort);
30
31         Registry registry = LocateRegistry.createRegistry(bindingPort);
32
33         registry.rebind(serviceName, remoteObjectStub);
```

- Assegnamento nome host con il metodo `setProperty()`
- Esportazione dell'oggetto con `exportObject()`
- Creazione del registro con `createRegistry()`
- Bind del servizio su RMI registry con il metodo `rebind()`

SERVER RMI (5)

```
36     Object alive=new Object();
37     synchronized (alive) {
38         try{
39             alive.wait();
40         } catch(InterruptedException e) {}
41     }
```

- Creazione di un oggetto di classe Object che rimane in attesa e impedisce la terminazione del processo server da parte di JVM nei periodi di lunga inattività (Non necessario se eseguito in locale).

CLIENT: RMI E JSP

Il client dovrà esporre un'interfaccia all'utente per visualizzare lo stato delle luci e permettergli di modificarlo. L'interfaccia sarà generata da una pagina JSP, mentre la parte relativa alla comunicazione con il server RMI sarà implementata dalla classe *Client.java*.

CLIENT RMI (1)

```
9 public class Client {
10     private RemoteDomusInterface rdi;
11
12     public Client() {
13         try {
14             String serviceName = "Domus";
15
16             Registry registry = LocateRegistry.getRegistry("<hostname>");
17
18             rdi = (RemoteDomusInterface) registry.lookup(serviceName);
19         } catch (RemoteException e) {
20             e.printStackTrace();
21         } catch (Exception e) {
22             e.printStackTrace();
23         }
24     }
```

- Utilizzo del metodo `getRegistry()` per ottenere un riferimento del registro
- Interrogazione del registro per ottenere un riferimento all'oggetto remoto con il metodo `lookup()`

CLIENT RMI (3)

```
26  public boolean getLightState(int room) {
27      try {
28          return rdi.getLightState(room);
29      } catch (RemoteException e) {
30          e.printStackTrace();
31      }
32      return false;
33  }
34
35  public void setLightState(int room, boolean b){
36      try {
37          rdi.setLightState(room, b);
38      } catch (RemoteException e) {
39          e.printStackTrace();
40      }
41  }
```

Il client utilizza i metodi remoti come se fossero locali attraverso il riferimento all'oggetto remoto *rdi*. L'unico accorgimento è intercettare eventuali *RemoteException* nel caso in cui si verifichino degli errori di rete (e.g. il server non è disponibile).

La pagina *index.jsp* si occupa di istanziare un oggetto di tipo *Client*, con cui potrà effettuare le invocazioni dei metodi remoti, e di presentare all'utente un'interfaccia grafica. Java Server Pages è una tecnologia che permette la creazione di pagine web dinamiche attraverso l'uso di specifici tag. Le pagine JSP dovranno essere eseguite all'interno di un application server (e.g. Apache Tomcat)

Creazione del Bean con cui si effettueranno le invocazioni a metodi remoti (*index.jsp*)

```
28 <body>
29   <jsp:useBean id="remote_object" scope="session"
30     class="domus.client.Client" />
```


Visualizzazione dello stato delle luci, chiamando il metodo remoto *getLightState()* per ogni stanza.

```
51     out.print("<a href=\"setLight.jsp?state=off&room=0\">
52         <span class=\"label label-success\">on</span>
53     </a>");
54 else
55     out.print("<a href=\"setLight.jsp?state=on&room=0\">
56         <span class=\"label label-danger\">off</span>
57     </a>");
58 %>
59     </div>
60 </div>
```

Lettura del form e invocazione del metodo remoto
setLightState()

```
11 <jsp:useBean id="remote_object" scope="session"
12     class="domus.client.Client" />
13 <%
14     try {
15         String state = request.getParameter("state");
16         String tmpRoom = request.getParameter("room");
17         int room = Integer.parseInt(tmpRoom);
18
19         if(state.equals("on")) {
20             remote_object.setLightState(room, true);
21         } else {
22             remote_object.setLightState(room, false);
23         }
24
25     } catch (Exception e) { }
26     response.sendRedirect("index.jsp");
27 %>
```

DEPLOYMENT DELLA DEMO

Questo semplice esempio può essere riprodotto in locale eseguendo il main di *Server.java* e la pagina *index.jsp* (per cui sarà necessaria l'installazione di Apache Tomcat).

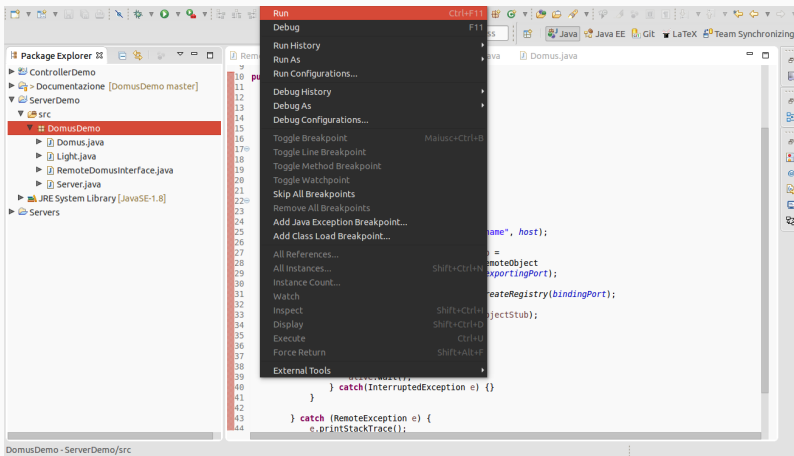
Di seguito troverete le istruzioni per eseguire la demo usando *eclipse*. Troverete il codice sorgente nel repository RMI_Demo disponibile su https://github.com/DomusDemo/RMI_Demo.

In eclipse, creare un nuovo progetto (*file > new > project*, selezionare *Java > Java project*, inserire il nome del progetto e cliccare su *finish*) e creare

- il package *DomusDemo*
- l'interfaccia *RemoteDomusInterface*
- le classi *Domus*, *Light* e *Server*.

ESEGUIRE IL SERVER (2)

Una volta create le classi, potete eseguire il main della classe *Server* (Dal menù selezionare *run > run*).



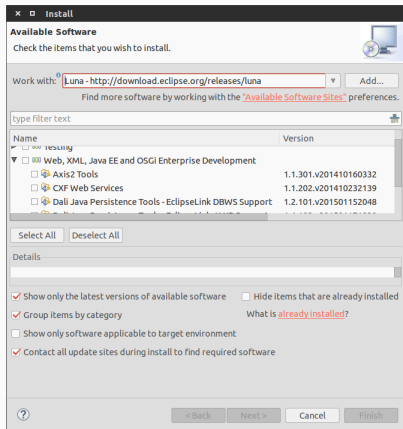
Per eseguire il controller si avrà bisogno di

- installare *Web Tool Platform* in eclipse e creare un *Dynamic Web Project*
- configurare il server *Tomcat* in eclipse

Dal menù di eclipse, andare su *help > install new software....*
Sotto la categoria *Web, XML, Java EE and OSGI Enterprise Development* installare

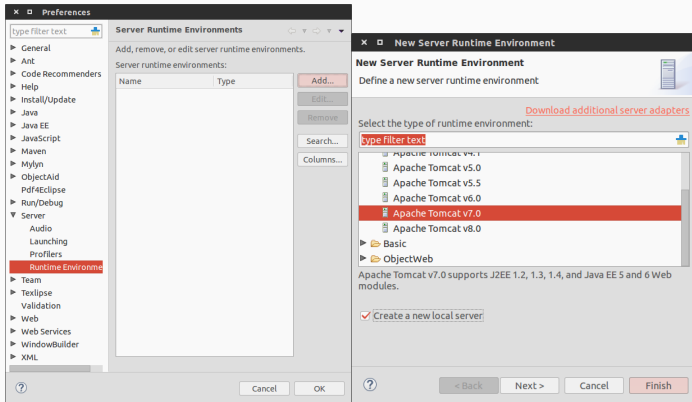
- Eclipse Java EE Developer Tools
- Eclipse Java Web Developer Tools
- Eclipse Web Developer Tools
- JST Server Adapters
- JST Server Adapters Extensions

ESEGUIRE IL CONTROLLER (3) - INSTALLARE WPT



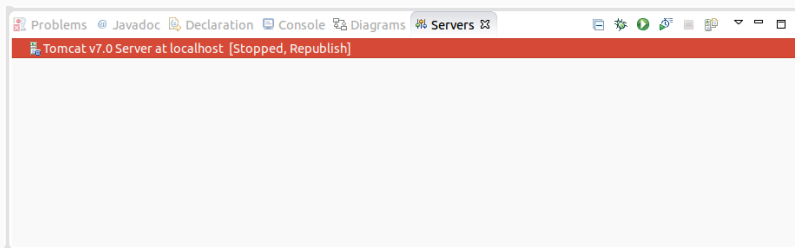
ESEGUIRE IL CONTROLLER (4) - CONFIGURARE WPT

Per configurare WPT, selezionare *window > preferences > server > runtime environments*, cliccare su *add* e selezionare *Apache Tomcat v7.0* (spuntare *create a new local server*). Scegliere la directory locale dove installare Tomcat e cliccare su *finish*.



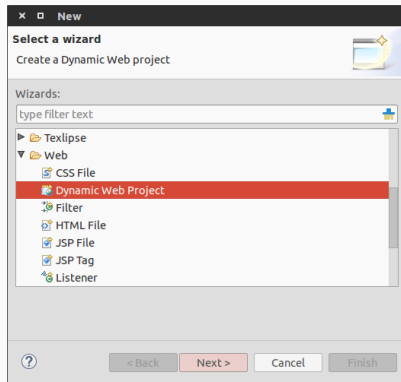
ESEGUIRE IL CONTROLLER (5) - CONFIGURARE TOMCAT

Per eseguire Tomcat è utile aprire la view server. Selezionare *window > show view > other... > server > show servers*. Da questa view, si potrà eseguire e fermare Tomcat.



ESEGUIRE IL CONTROLLER (6) - CREARE IL DYNAMIC WEB PROJECT

Per creare il progetto del controller, dal menù *file > new > other* selezionare *Web > Dynamic Web Project*



ESEGUIRE IL CONTROLLER (7) - CREARE IL DYNAMIC WEB PROJECT

Immettere il nome del progetto e cliccare su *next*, di nuovo *next* nella finestra successiva, e infine spuntare la voce *Generate web.xml deployment descriptor* per poi cliccare su *finish*

New Dynamic Web Project

Dynamic Web Project
Create a standalone Dynamic Web project or add it to a new or existing Enterprise Application.

Project name:

Project location:
☒ Use default location
Location:

Target runtime:

Dynamic web module version:

Configuration:

A good starting point for working with Apache Tomcat v7.0 runtime. Additional facets can later be installed to add new functionality to the project.

EAR membership:
☐ Add project to an EAR
EAR project name:

Working sets:
☐ Add project to working sets

New Dynamic Web Project

Web Module
Configure web module settings.

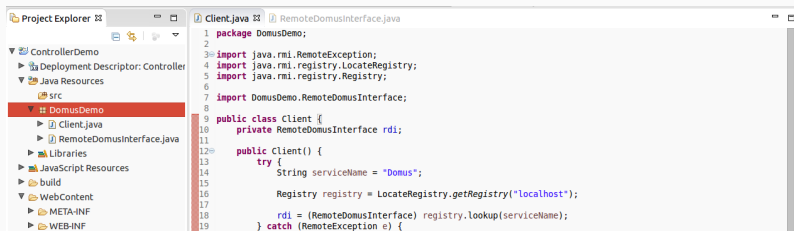
Context root:

Content directory:

☒ Generate web.xml deployment descriptor

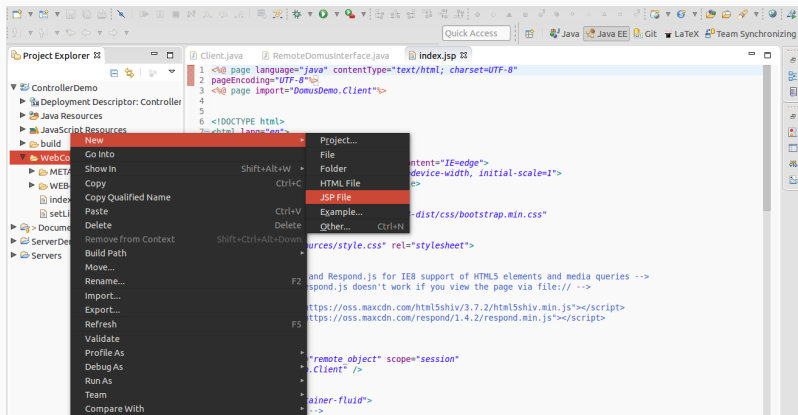
ESEGUIRE IL CONTROLLER (8) - CREARE LE CLASSI

Creare un nuovo package *DomusDemo* ed al suo interno l'interfaccia *RemoteDomusInterface* e la classe *Client*



ESEGUIRE IL CONTROLLER (9) - CREARE LE PAGINE JSP

All'interno della cartella *WebContent*, creare le pagine JSP *index.jsp* e *setLight.jsp*. Per farlo, cliccare con il destro sulla cartella e selezionare *new > JSP file*. Inserire il nome del file e cliccare su *finish*.

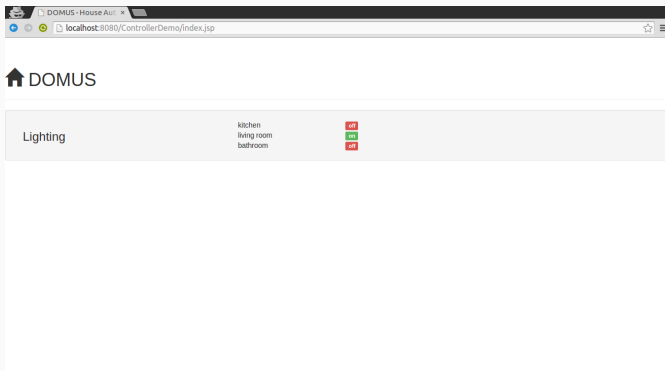


ESEGUIRE IL CONTROLLER (10) - CREARE LE PAGINE JSP

Una volta accertato che il server RMI (il main di *Server.java*) sia in esecuzione, si può eseguire il controller.

A questo punto, aprire il browser e inserire l'URL

http://localhost:8080/ControllerDemo/index.jsp (può variare in base al nome che si è dato al progetto).



DOMANDE?