Aplicatii Integrate pentru Intreprinderi
Semestrul de Toamna 2013

Laborator 5

Proiectarea de aplicatii distribuite folosind tehnologia RMI

Continut

- **Ce este RMI?**
- **Arhitectura mecanismului RMI**
- **Notiunea de obiect/clasa serializabil(a)**
- **Clase si interfete "la distanta"**
- **Dezvoltarea serverului RMI**
- **Dezvoltarea clientului RMI**

Ce este RMI?

- RMI – Remote Method Invocation
- extinde functionalitatea RPC (Remote Procedure Call) pe care o implementeaza la nivel obiect
- comunicatia la distanta intre programe scrise in Java
  - invocarea metodelor unui obiect existent in alt spatiu de adrese
    - pe aceeasi masina
    - pe masini diferite
  - dezvoltarea aplicatiilor distribuite in acelasi mod in care ar fi dezvoltate aplicatiile nedistribuite

Folosirea obiectelor locale si "la distanta" in Java

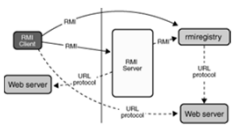
Concept	Obiect local	Obiect „la distanță"
definirea obiectului	se face într-o clasă Java	definirea comportamentului (exportat) al obiectului „la distanță" se face printr-o interfață care trebuie să extindă interfața `java.rmi.Remote`
implementarea obiectului	se face în clasa Java corespunzătoare	comportamentul obiectului accesat „la distanță" este realizat printr-o clasă care implementează interfața
crearea obiectului	instanța unui obiect nou se face prin operatorul **new**	o instanță nouă pentru un obiect aflat „la distanță" se face folosind operatorul **new** pe mașina gazdă (clientul nu poate crea direct obiectul „la distanță")
accesul obiectului	se realizează direct printr-o variabilă de tip referință	un obiect „la distanță" este accesat printr-o referință care indică spre un delegat al implementării interfeței
referințe	o referință indică direct spre obiectul din heap corespunzător	o referință „la distanță" indică spre un obiect delegat (stub) alocat local în heap; obiectul delegat conține informații ce permit conectarea la obiectul „la distanță" unde este realizată implementarea metodelor
referințe active	un obiect este considerat „activ" dacă există cel puțin o referință către el	într-un mediu distribuit unde mașinile virtuale pot manifesta erori critice sau conexiunea poate fi pierdută, o referință este activă pentru un obiect la distanță dacă accesarea se face într-o anumită perioadă de timp (de încriere); dacă pentru un obiect s-a renunțat (în mod explicit) la toate referințele sau toate referințele au expirat, obiectul „la distanță" este disponibil pentru colectarea memoriei (emg. garbage collection) distribuită.
finalizarea	metoda `finalize()` (în caz că este definită) se apelează înainte ca memoria aferentă obiectului să fie dezalocată (prin garbage collector)	dacă obiectul „la distanță" implementează interfața **Unreferenced**, metoda definită de această interfață este apelată când referințele la obiect sunt distruse
colectarea memoriei disponibile	un obiect spre care nu mai există referințe (locale) devine candidat pentru mecanismul de colectare a memoriei disponibile (pentru dezalocarea memoriei aferente)	există modul de colectare a memoriei distribuit care lucrează cu modulul garbage collector local, apelându-se atunci când nu există referințe realizate „la distanță" și toate referințele locale pentru obiectul accesat „la distanță" au fost distruse
excepții	un program trebuie să trateze toate excepțiile (runtime sau alt tip)	pentru asigurarea robusteții aplicațiilor distribuite, programele trebuie să trateze excepțiile `RemoteException` care ar putea fi aruncate

Arhitectura mecanismului RMI

- model de obiecte distribuite
  - integrat in limbajul de programare
  - compatibil cu modelul de obiecte locale
- 2 programe separate – comunicare in ambele sensuri
  - **server**
    - ❶ creeaza obiectele la distanta
    - ❷ "publica" referintele catre obiecte
    - ❸ asteapta clientii sa invoce metodele implementate
  - **client**
    - ❶ obtine o referinta spre unul sau mai multe obiecte existente pe server
    - ❷ invoca metodele specifice acestor obiecte

Arhitectura mecanismului RMI (cont'd)

- **localizarea obiectelor "la distanta"** – serviciul de nume rmiregistry
- **comunicarea cu obiectele "la distanta"** – similar ca in cazul local
- **incarcarea definitiilor de clase pentru obiectele ale caror metode sunt invocate** – concomitent cu transmiterea datelor

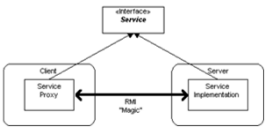


Modul de functionare

- obtinerea definitiei clasei
  - transmiterea se face prin clasa – comportament consistent intre mai multe masini virtuale diferite
  - comportamentul aplicatiei este imbogatit prin functionalitati implementate pe o alta masina virtuala
- obiecte "la distanta" – obiecte invocate intre masini virtuale diferite
  - implementeaza o interfata ce extinde `java.rmi.Remote`
  - fiecare metoda a interfetei genereaza cel putin exceptia `java.rmi.RemoteException`

Modul de functionare (cont'd)

- obiectele "la distanta"
  - serverul "publica" la serviciul de nume rmiregistry un ciot (*eng.* stub) care este transmis la cerere clientului
    - implementeaza acelasi set de interfete ca si obiectul "la distanta"
    - poate fi convertit la oricare din interfetele obiectului aflat la distanta
  - pe client, acesta joaca rolul de reprezentant (*eng.* proxy)
    - metodele "la distanta" sunt apelate pe acesta
    - acesta este responsabil pentru executia metodei invocate



Etapele dezvoltarii unei aplicatii distribuite folosind RMI

- ❶ proiectarea si implementarea componentelor aplicatiei distribuite
  - definirea interfetelor la distanta (metodele ce pot fi apelate "la distanta")
  - implementarea obiectelor accesibile "la distanta" si a tipurilor de date folosite ca parametri si/sau valori intoarse
  - implementarea clientului
- ❷ compilarea surselor
- ❸ "publicarea" claselor pentru a fi accesibile prin retea (de regula, printr-un server web)
- ❹ rularea componentelor aplicatiei distribuite
  - serviciul de nume RMI (rmiregistry)
  - serverul
  - clientul

Notiunea de obiect (clasa) serializabil(a)

- tipuri de clase folosite in mecanismul RMI
  - clase `Remote` – ale caror instante pot fi accesate
    - local – in spatiul de adrese in care au fost create
    - "la distanta" – in alte spatii de adrese (masini virtuale) prin intermediul unui delegat, ce impune unele limitari comparativ cu utilizarea locala
  - clase `Serializable` – ale caror instante pot fi copiate intre spatii de adrese
- serializarea – folosita in RMI pentru transmiterea obiectelor **prin valoare** intre masini virtuale diferite
  - pentru parametri metodelor "la distanta" si valorile intoarse de acestea care se copiaza intre spatiile la adrese

Notiunea de obiect (clasa) serializabil(a) – cont'd

- o clasa serializabila
  - implementeaza interfata `java.io.Serializable`
    - nu are atribute sau metode
    - serveste pentru a defini semantica de a fi serializabil
    - serializarea se realizeaza intern, pentru orice camp al clasei care nu este marcat ca `static` sau `transient`
  - atributul `serialVersionUID` este folosit pentru a verifica daca o clasa incarcata corespunde unui obiect serializabil (in caz contrar se genereaza `InvalidClassException`)
    - in cazul ca nu este definita, o valoare implicita calculata pe baza proprietatilor clasei va fi utilizata
    - valoarea implicita depinde de implementari ale compilatorului
    - `ANY-ACCESS-MODIFIER static final long serialVersionUID = …L;`

Notiunea de obiect (clasa) serializabil(a) – cont'd

- clasele standard, a caror stare are semnificatie in masini virtuale diferite, sunt de regula serializabile
  - clasa `Thread` nu este serializabila, pentru ca starea ei este strict legata de contextul masinii virtuale in care ruleaza
- comportamentele cu privire la procesul de serializare-deserializare pot fi specificate explicit
  - `private void writeObject(java.io.ObjectOutputStream out) throws IOException`
  - `private void readObject(java.io.ObjectInputStream in) throws IOException, ClassNotFoundException`
  - `private void readObjectNoData() throws ObjectStreamException`
- campurile unei clase serializabile trebuie sa fie ele insele serializabile !!!

Clase si interfete "la distanta"

```
import java.rmi.Remote;
import java.rmi.RemoteException;
import java.util.ArrayList;

public interface Reservation extends Remote {
public ArrayList<Interval> getTimetable() throws RemoteException;
public int getAvailableSeats(Interval interval)
    throws RemoteException;
public boolean makeReservation(int clientId, Interval interval,
    int numberOfSeats) throws RemoteException;
public boolean cancelReservation(int clientId, Interval interval)
    throws RemoteException;
}
```

Clase si interfete "la distanta" (cont'd)

- extinderea interfetei `java.rmi.Remote` – metodele componente vor putea fi apelate din alta masina virtuala Java
  - metodele accesibile "la distanta" trebuie sa genereze exceptia `java.rmi.RemoteException`
    - o astfel de exceptie denota faptul ca s-au produs erori de comunicare sau de protocol
  - obiectele ce reprezinta instante ale claselor care implementeaza aceasta interfata sunt considerate obiecte "la distanta"

Clase serializabile – exemplu

```
import java.io.Serializable;
import java.util.GregorianCalendar;

public class Interval implements Serializable {
    private static final long serialVersionUID = 1024L;
    GregorianCalendar startingTime;
    GregorianCalendar endingTime;
    public Interval() {
        startingTime = new GregorianCalendar();
        endingTime = new GregorianCalendar();
    }
    public Interval(GregorianCalendar startingTime, GregorianCalendar endingTime) {
        this.startingTime = startingTime;
        this.endingTime = endingTime;
    }
    public void setStartingTime(GregorianCalendar startingTime) {
        this.startingTime = startingTime;
    }
    public GregorianCalendar getStartingTime() {
        return startingTime;
    }
    public void setEndingTime(GregorianCalendar endingTime) {
        this.endingTime = endingTime;
    }
    public GregorianCalendar getEndingTime() {
        return endingTime;
    }
}
```

Incarcarea claselor pe server si pe client

- interfetele "la distanta" si clasele folosite pentru parametri si valori intoarse trebuie sa fie cunoscute
  - pe server – furnizeaza implementarile lor
  - pe client – le invoca functionalitatea
- frecvent, clasele ce trebuie incarcate atat pe server cat si pe client sunt impachetate intr-o arhiva .jar
  - ❶ compilarea claselor → `javac *.java`
  - ❷ impachetarea claselor → `jar cvf *.class`
  - !!! in cazul in care clasele fac parte dintr-un package, ele trebuie sa se gaseasca intr-o structura de directoare echivalenta cu numele pachetului din care fac parte

Dezvoltarea serverului RMI

- implementeaza interfata "la distanta"
  - poate implementa si alte metode, accesibile local
    - constructorul clasei
    - metoda `main`
  - parametrii / valorile intoarse ale metodelor la distanta pot fi
    - obiecte serializabile
    - obiecte "la distanta"
    - tipuri primitive de date
    - NU – fire de executie, descriptori de fisier
- obiectele serializabile sunt transmise prin valoare, realizandu-se copii ale acestora
- obiectele la distanta sunt transmise prin referinta (un ciot este delegate la nivelul clientului si implementeaza metodele accesibile la distanta)
  - modificarile realizate asupra ciot-ului delegate sunt vizibile in obiectul de pe server

Dezvoltarea serverului RMI (2) – exemplu

```
import java.rmi.RemoteException;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import java.rmi.server.UnicastRemoteObject;
import java.util.ArrayList;
public class ReservationServer implements Reservation {
    public ReservationServer() { }
    public ArrayList<Interval> getTimetable() throws RemoteException {
        ...
    }
    public int getAvailableSeats(Interval interval) throws RemoteException {
        ...
    }
    public boolean makeReservation(int customerId, Interval interval, int numberOfSeats) throws RemoteException {
        ...
    }
    public boolean cancelReservation(int customerId, Interval interval) throws RemoteException {
        ...
    }
    public static void main (String[] args) {
        if (System.getSecurityManager() == null) {
            System.setSecurityManager(new SecurityManager());
        }
        try {
            String serviceName = "ReservationService";
            Reservation service = new ReservationServer();
            Reservation proxy =
            (Reservation)UnicastRemoteObject.exportObject(service, 0);
            Registry registry = LocateRegistry.getRegistry();
            registry.rebind(serviceName, proxy);
        } catch (Exception exception) { }
    }
}
```

Dezvoltarea serverului RMI (3)

- ❶ crearea si instalarea unui sistem de securitate
  - protejeaza accesul la resurse pentru codul descarcat care va rula in masina virtuala (acces la sistemul local de fisiere, alte operatii privilegiate)

```
if (System.getSecurityManager() == null) {
    System.setSecurityManager(new SecurityManager());
}
```

  - specificat de proprietatea `-Djava.security.policy`
- ❷ se creeaza o instanta a clasei "la distanta" si delegatul care va fi publicat catre serviciul de nume rmiregistry

```
Reservation service = new ReservationServer();
Reservation proxy = Reservation)UnicastRemoteObject.exportObject(service, 0);
```

Dezvoltarea serverului RMI (4)

- ❸ localizarea serviciului de nume RMI (rmiregistry)
  - interfata `java.rmi.registry.Registry`
  - functionalitati
    - inregistrarea obiectelor "la distanta" → `rebind`
    - cautarea obiectelor "la distanta" → `lookup`
  - interfata `java.rmi.registry.LocateRegistry`
    - metode statice pentru obtinerea referintelor la serviciul de nume
    - `getRegistry` – accepta ca parametri adresa masinii pe care ruleaza serviciul de nume si portul (implicit – localhost / 1099)

```
Registry registry = LocateRegistry.getRegistry();
```

- ❹ inregistrarea obiectului "la distanta" (delegatului) la serviciul de nume
  - `rebind` – metoda la distanta catre serviciul de nume RMI (rmiregistry)

```
registry.rebind("ReservationService", proxy);
```

Dezvoltarea serverului RMI (5)

- metoda `main` nu se incheie atata vreme cat exista referinte catre obiectele existente "la distanta"
  - de catre serviciul de nume RMI (rmiregistry)
  - de catre clienti care ii acceseaza metodele prin intermediul serviciului de nume RMI (rmiregistry)
- exceptia de tip `RemoteException` poate fi generata de
  - `UnicastRemoteObject.exportObject`
  - `rebind`
- metoda `rebind` nu este blocanta, in continuare se pot realiza si alte operatii pe server in metoda `main`

Rularea serverului RMI

- ❶ pornirea serviciului de nume RMI (rmiregistry)

```
> start rmiregistry
-J-Djava.rmi.server.useCodeBaseOnly=false [port]
```

```
# rmiregistry
-J-Djava.rmi.server.useCodeBaseOnly=false [port] &
```

- calea catre Java trebuie sa existe in variabila de mediu classpath

!!! verificati versiunea de Java din variabila de mediu classpath; aceasta trebuie sa fie aceeasi cu cea cu care dezvoltati proiectul si cu care au fost impachetate clasele

- ❷ specificarea politicii de securitate

```
grant codebase "file://<cale_absoluta>\\ReservationServer\\-" {
    permission java.security.AllPermission;
};
```

Politici de securitate in Java

- sintaxa

```
grant [codebase CodeBase] {
    permission Permission;
    permission Permission;
    permission Permission;
  };
```

unde

- CodeBase – URL specificand locatia claselor asupra carora se aplica politica de securitate
  - poate sa nu fie specificat, se aplica asupra tuturor claselor din classpath
  - *.jar → doar clasele din arhiva jar specificata
  - / → doar clasele din directorul specificat
  - * → toate clasele si arhivele jar din directorul specificat
  - – → toate clasele si arhivele jar din directorul specificat si toate subdirectoarele sale
  - exemple: file:${application), file:${jars), file:${webComponent}, file:${connectorComponent}
- Permission
  - tipul permisiunii: java.security.SecurityPermission, java.io.FilePermission, java.util.PropertyPermission, java.net.SocketPermission
  - obiectul asupra caruia se aplica politica de securitate: "${user.install.root}${/}bin${/}DefaultDB${/}*", "localhost:1024-"
  - actiunile permise: read, write, delete, listen

Rularea serverului RMI (2)

- ❸ compilarea surselor de pe server
- ❹ rularea serverului

```
> java -cp .[;|:]reservation.jar
       -Djava.rmi.server.codebase=file:///<cale_absoluta>\reservation.jar
       -Djava.rmi.server.hostname=localhost
       -Djava.security.policy=policy
       ReservationServer
```

- `java.rmi.server.codebase` → locatia de unde pot fi descarcate definitii pentru clasele provenind de la server
- `java.rmi.server.hostname` → numele masinii / adresa ce urmeaza a fi completata in obiectele de tip ciot care v or fi exportate, aceasta fiind folosita de clienti cand vor incerca sa apeleze metodele la distanta
  - implicit, se foloseste adresa IP indicata de `java.net.InetAddress.getLocalHost`
- `java.security.policy` → fisierul continand politica de securitate ce indica permisiunile care vor fi acordate

Dezvoltarea clientului RMI

- ❶ crearea si instalarea unui sistem de securitate
  - in procesul de obtinere a referintei catre obiectul ciot corespunzator obiectului la distanta se poate intampla sa se descarce definitii de clase de la server
- ❷ localizarea serviciului de nume RMI (rmiregistry)
- ❸ obtinerea unei referinte catre obiectul "la distanta" de la serviciul de nume RMI (rmiregistry) prin indicarea numelui care il identifica

```
Registry registry = LocateRegistry.getRegistry(args[0]);
Reservation reservation = (Reservation)registry.lookup(serviceName);
```

- ❹ apelurile de metode la distanta se realizeaza similar ca in cazul local

```
boolean result =
    reservation.makeReservation(customerId,interval,numberOfSeats);
```

Rularea clientului RMI

- ❶ pornirea serviciului de nume RMI (rmiregistry)
- ❷ specificarea politicii de securitate

```
grant codebase "file://<cale_absoluta>\\ReservationClient\\-" {
    permission java.security.AllPermission;
};
```

- ❸ compilarea surselor de pe client
- ❹ rularea clientului

```
> java -cp .[;|:]reservation.jar
       -Djava.rmi.server.codebase=file:///<cale_absoluta>\
       -Djava.security.policy=policy
       ReservationClient <adresa_server>
```