Κατανεμημένα Συστήματα
7ο Φυλλάδιο εργαστηρίου (04/05/2017)

Java Remote Method Invocation (RMI)

Κατασκευή του πελάτη

Η εφαρμογή του πελάτη είναι αυτή που αιτείται την εκτέλεση μιας υπηρεσίας από τον εξυπηρετητή, παρέχοντας σε αυτόν τις κατάλληλες και απαιτούμενες, για την εκτέλεση, παραμέτρους.

Τα βασικά στοιχεία που πρέπει να περιλαμβάνονται σ' ένα πρόγραμμα πελάτη, είναι τα εξής:

- Πρέπει να ορίζεται μια αναφορά, ο τύπος της οποίας πρέπει να είναι η RMI διεπαφή, μέσω της οποίας θα γίνεται κλήση στον εξυπηρετητή για την εκτέλεση της επιθυμητής μεθόδου (γραμμές 5-6).
- Πρέπει να ζητείται, η εκτέλεση των επιθυμητών μέθοδών και να εισάγονται τα απαιτούμενα ορίσματα για τις μεθόδους αυτές (γραμμές 7-8).
- Προαιρετικά στο πρόγραμμα του πελάτη μπορεί να εγκατασταθεί και ένας διαχειριστής ασφάλειας.

Στο παράδειγμά μας έχουμε δημιουργήσει τον πελάτη με το όνομα `OperationClient` και είναι αποθηκευμένος στο αρχείο με όνομα `OperationClient.java`.

```
// OperationClient.java
1. import java.rmi.*;
2. public class OperationClient {
3. public static void main(String args[]) {
4. try {
5. String name = "//localhost/OperationServer";
6. Operations look_op =(Operations) Naming.lookup(name);
7. look_op.setNum(15,20);
8. int result=look_op.sum();
9. System.out.println(result);
10.}
11. catch (Exception e) {
12.    System.out.println("OperationClient err: " + e);
13.    System.exit(1);
14.    }
15. }
16. }
```

Στην εφαρμογή του πελάτη διακρίνουμε τα εξής στοιχεία:

Στην γραμμή 5 αναφέρεται η διεύθυνση αντικειμένου από την οποία θα αναζητηθεί η επιθυμητή υπηρεσία.

Στην γραμμή 6 δημιουργείται το αντικείμενο, το οποίο είναι του ιδίου τύπου με την διεπαφή, που θα χρησιμοποιήσουμε για να αναφερθούμε προς το απομακρυσμένο

αντικείμενο το οποίο υλοποιεί την διεπαφή. Η αναζήτηση πραγματοποιείται βάσει της διεύθυνσης που έχουμε δηλώσει στην γραμμή 5.

Στις γραμμές 7 και 8 καλούνται, προς εκτέλεση, οι επιθυμητές μέθοδοι σα να ήταν στην ίδια εφαρμογή με τον πελάτη.

Στο σημείο αυτό θα πρέπει να τονίσουμε, όπως αναφέραμε, την δυνατότητα να δημιουργήσουμε και να εγκαταστήσουμε έναν διαχειριστή ασφάλειας, ανάλογο με αυτόν που δημιουργούμε στον εξυπηρετητή και στην εφαρμογή του πελάτη. Γενικότερα, όπως είδαμε και στον εξυπηρετητή, το μοντέλο RMI μας δίνει την δυνατότητα δημιουργίας ενός ασφαλούς περιβάλλοντος για την εφαρμογή μας αλλά δεν αποτελεί και προϋπόθεση προκειμένου να δημιουργηθεί μια εφαρμογή πελάτη-εξυπηρετητή βασιζόμενη στην τεχνολογία RMI.

Εκτέλεση της εφαρμογής

Πιο πάνω παρακολουθήσαμε τα προαπαιτούμενα εκείνα χαρακτηριστικά που πρέπει να υιοθετούν τα τμήματα μιας εφαρμογής πελατη-εξυπηρετητή βάσει του μοντέλου RMI. Όπως γνωρίζουμε για να πραγματοποιηθεί μια επικοινωνία πελάτη-εξυπηρετητή σε ένα κατανεμημένο και ετερογενές περιβάλλον απαιτείται η χρήση ειδικών τμημάτων κώδικα τα οποία έχουν ως σκοπό να μεταφράζουν τις μεταδιδόμενες πληροφορίες σε μορφή τέτοια η οποία θα είναι ανεξάρτητη του ετερογενούς περιβάλλοντος και κατανοητή από τον αποδέκτη. Τα τμήματα αυτά δεν είναι άλλα από τα λεγόμενα stub, από την πλευρά του πελάτη, και skeleton, από την πλευρά του εξυπηρετητή. Υπεύθυνος για την δημιουργία αυτών των τμημάτων είναι ένας ειδικός μεταφραστής της Java, ο rmic.

Μετά την ολοκλήρωση της δημιουργίας του κώδικα που αφορά τον εξυπηρετητή έχουμε δύο αρχεία της μορφής interface_name.java και server_name.java (στο παράδειγμά μας έχουμε αντίστοιχα τα Operations.java και OperationServer.java). Μεταγλωττίζουμε τα αρχεία αυτά με τη βοήθεια του μεταφραστή της Java, **javac**, με αποτέλεσμα την δημιουργία των αρχείων κλάσεων της μορφής interface_name.class και server_name.class (στο παράδειγμά μας Operations.class και OperationServer.class). Ακολούθως εφαρμόζουμε τον μεταφραστή **rmic** πάνω στο αρχείο κλάσης του εξυπηρετητή server_name.class (OperationServer.class) με αποτέλεσμα την δημιουργία των επιθυμητών τμημάτων server_name_stub.class και server_name_skeleton.class (OperationServer_Stub.class και OperationServer_Skel.class).

Μετά την δημιουργία όλων των απαιτούμενων κλάσεων (interface, server, client, stub και skeleton) πρέπει να μεταφέρουμε τα αρχεία κλάσεων των interface, client και stub, στην περιοχή απ' όπου θέλουμε να τρέξουμε την εφαρμογή του πελάτη (μπορεί να είναι στον ίδιο υπολογιστή ή σε απομακρυσμένο) και τα αρχεία κλάσεων των interface, server, stub και skeleton, στην διεύθυνση απ' όπου έχουμε δηλώσει ότι θα τρέχει η εφαρμογή του εξυπηρετητή.

Τα τελευταία βήματα που έχουμε να κάνουμε προκειμένου να εκτελέσουμε την εφαρμογή μας είναι:

- Έναρξη λειτουργίας του καταχωρητή (registry) στον οποίο θα γίνεται η καταχώρηση του αναφορικού ονόματος του απομακρυσμένου αντικειμένου, έτσι ώστε ο κάθε υποψήφιος πελάτης να μπορεί να αποκτήσει την αναφορά (reference)

προς το επιθυμητό απομακρυσμένο αντικείμενο. Η έναρξη της λειτουργίας καταχώρησης υλοποιείται με την εντολή **start rmiregistry**.

- Έναρξη της λειτουργίας της εφαρμογής του εξυπηρετητή με την εντολή **java server_name**. Με την έναρξη λειτουργίας του εξυπηρετητή πραγματοποιείται η καταχώρηση του αναφορικού ονόματος του απομακρυσμένου αντικειμένου, το οποίο και αναμένει για τυχόν κλήσεις από πιθανούς πελάτες.

- Τέλος, έναρξη της λειτουργίας της εφαρμογής του πελάτη με την εντολή **java client_name**. Με την έναρξη της λειτουργίας του πελάτη, αναζητείται το απομακρυσμένο, επιθυμητό, αντικείμενο βάσει της αναφορικής του ονομασίας και ξεκινάει η επικοινωνία, μέσω των stub και skeleton, του πελάτη με τον εξυπηρετητή.

RMI ΣΥΓΧΡΟΝΙΣΜΟΣ – synchronized, wait, notify

Σε ένα κατανεμημένο μοντέλο RMI είναι πολύ πιθανόν να είναι αναγκαίος ο συγχρονισμός στις εργασίες που επιτελούνται. Το πρόβλημα του συγχρονισμού διεργασιών είναι εξαιρετικά δύσκολο, αφού εκτός από το να παρέχει τη δυνατότητα αποκλειστικής χρήσης στα αντικείμενα, ένας μηχανισμός συγχρονισμού πρέπει να εξασφαλίζει την αποφυγή αδιεξόδων.

Ένας τρόπος συγχρονισμού σε κατανεμημένα μοντέλα με μηχανισμούς RMI, είναι με τη χρήση της ***synchronized*** σε μεθόδους ή σε τμήματα κώδικα, καθώς και δυνατότητα συγχρονισμού με τις μεθόδους ***wait*** και ***notify*** της κλάσης Object. Οι μηχανισμοί αυτοί βασίζονται στην έννοια του κλειδώματος (lock). Κάθε αντικείμενο της Java μπορεί να θεωρηθεί ότι διαθέτει ένα κλειδί. Το ίδιο συμβαίνει και για κάθε κλάση. Προκειμένου να επιτευχθεί αποκλειστική πρόσβαση σε ένα σύνολο μεθόδων ενός αντικειμένου, αυτές δηλώνονται ως synchronized (συγχρονισμένες). Στη συνέχεια για να κληθεί μια συγχρονισμένη μέθοδος ενός αντικειμένου, πρέπει να αποκτηθεί το κλειδί του αντικειμένου από το νήμα εκτέλεσης που πραγματοποιεί την κλήση. Το κλειδί επιστρέφεται μετά την εκτέλεση της μεθόδου.

1η Εργαστηριακή Άσκηση – RMI Client

Στο προηγούμενο φυλλάδιο υλοποιήσατε έναν εξυπηρετητή με την χρήση RMI τεχνολογίας που παρέχει υπηρεσίες διαχείρισης τηλεφωνικού καταλόγου. Υλοποιήστε την εφαρμογή του πελάτη η οποία θα επικοινωνεί με τον εξυπηρετητή και θα χρησιμοποιεί τις λειτουργίες του.

Απάντηση

```
import java.rmi.*;
import java.net.MalformedURLException;
public class MainClass {
```

```
	public static void main(String[] args) {
	try {
	String name = "//localhost/PhoneDirectory";
	DirectoryOperations look_op
=(DirectoryOperations)Naming.lookup(name);
	Contact c = look_op.searchNumber("Pappas");
	System.out.println(c.getName()+"->"+c.getNumber());

	look_op.insertContact("Papadopoulos", "Vathi", "2273054321");

	c = look_op.searchNumber("Papadopoulos");
	System.out.println(c.getName()+"->"+c.getNumber());
	}
	catch (NotBoundException ex) {
			ex.printStackTrace();
	} catch (MalformedURLException ex) {
			 ex.printStackTrace();
	} catch (RemoteException ex) {
			 ex.printStackTrace();
	}

	}
}
```

2η Εργαστηριακή Άσκηση – RMI Client - Server

Υλοποιήστε μια απλοποιημένη εφαρμογή Chat με την χρήση RMI τεχνολογίας. Ο πελάτης καλώντας κατάλληλες μεθόδους από τον εξυπηρετητή θα έχει τη δυνατότητα να στείλει ένα μήνυμα (ως αντικείμενο) στον εξυπηρετητή και να ενημερωθεί για όλα τα μηνύματα που έχουν σταλεί από τον ίδιο και από άλλους πελάτες μέχρι εκείνη την στιγμή. Ο εξυπηρετητής δέχεται τα μηνύματα των πελατών και τα αποθηκεύει άμεσα σε αρχείο κειμένου. Για κάθε μήνυμα αποθηκεύει το όνομα του αποστολέα και την ώρα/ημέρα αποστολής και ενημερώνει όλους τους πελάτες για το περιεχόμενο του αρχείου.

Απάντηση

ChatInterface

```
import java.rmi.*;

public interface ChatInterface extends Remote{

    public void sendMessage(ChatMessage msg) throws RemoteException;
    public String update( ) throws RemoteException;

}
```

ChatMessage

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


public class ChatMessage implements Serializable{
    private String name;
    private String msg;
    private Date date;

 public ChatMessage (String name, String msg){
     this.name=name;
     this.msg=msg;
     this.date = new Date();
 }

 public String getName(){
  return this.name;

 }

  public String getMessage(){
  return this.msg;

 }

    public Date getDate(){
  return this.date;

 }

}
```

ChatRMIServer

```
import java.io.*;
import java.net.MalformedURLException;
import java.rmi.*;
import java.rmi.server.*;
import java.util.logging.Level;
import java.util.logging.Logger;


public class ChatRMIServer extends UnicastRemoteObject implements
ChatInterface{


    public ChatRMIServer()throws RemoteException{
        super();

    }
    public void sendMessage(ChatMessage msg) throws RemoteException {
        try {
            BufferedWriter file = new BufferedWriter(new
FileWriter("ChatMessages.txt",true));
            file.write(msg.getName()+" : "+msg.getDate()+" : "+
msg.getMessage()+"\n");
            file.close();

        } catch (IOException ex) {
            ex.printStackTrace();
        }


    }


    public String update() throws RemoteException {
        String str="";
        String tmp="";
        try {
            BufferedReader file = new BufferedReader(new
FileReader("ChatMessages.txt"));
            while((tmp=file.readLine()) != null)
                str=str+tmp+"\n";
            file.close();
        } catch (IOException ex) {
            ex.printStackTrace();
        }

        return str;
    }

    public static void main(String[] args) {
```

```
      // RMISecurityManager security = new RMISecurityManager();
      // System.setSecurityManager(security);

        ChatRMIServer server;
        try {
            server = new ChatRMIServer ();

//1099 is the port number
Registry r = java.rmi.registry.LocateRegistry.createRegistry(1099);
            Naming.rebind("//localhost/ChatRMI", server);
            System.out.println("Waiting new Messages");
        } catch (RemoteException ex) {
            ex.printStackTrace();
        } catch (MalformedURLException ex) {
           ex.printStackTrace();
        }
    }
}
```

ChatRMIClient

```
import java.net.MalformedURLException;
import java.rmi.*;
import java.util.logging.Level;
import java.util.logging.Logger;


public class ChatRMIClient {


    public static void main(String[] args) {

     //RMISecurityManager security = new RMISecurityManager();
     //System.setSecurityManager(security);
        try {
            ChatInterface look_op
=(ChatInterface)Naming.lookup("//localhost/ChatRMI");

            look_op.sendMessage(new ChatMessage("nikos", "Hello
World!!!!"));

            System.out.println(look_op.update());


        } catch (NotBoundException ex) {
            ex.printStackTrace();
        } catch (MalformedURLException ex) {
            ex.printStackTrace();
```

```
        } catch (RemoteException ex) {
            ex.printStackTrace();
        }
    }
}
```

3η Εργαστηριακή Άσκηση – RMI Client - Server

Υλοποιήστε έναν «time server» με την χρήση RMI τεχνολογίας. Ο πελάτης καλώντας κατάλληλη απομακρυσμένη μέθοδο θα έχει τη δυνατότητα να ενημερώνεται για την τρέχουσα ώρα από τον εξυπηρετητή.

Απάντηση

```
import java.rmi.*;
public interface TimeOperations extends Remote{
    public String showTime() throws RemoteException;
}
```

```
import java.rmi.*;
import java.rmi.registry.Registry;
import java.rmi.server.*;
import java.util.logging.Level;
import java.util.logging.Logger;


public class TimeServer {
    public static void main(String[] args) {
        try {
            TimeServerImpl tsi= new TimeServerImpl();

Registry r = java.rmi.registry.LocateRegistry.createRegistry(1099);

            Naming.rebind("//localhost/time", tsi);
            System.out.println("Time Server is running ...");
        } catch (Exception ex) {
            ex.printStackTrace();
        }
    }
}
```

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

public class TimeServerImpl extends UnicastRemoteObject implements
TimeOperations{
```

```
    public TimeServerImpl() throws Exception {
        super();
    }

    public String showTime() throws RemoteException {

    Calendar calendar = new GregorianCalendar();

    String amORpm;
    int hour = calendar.get(Calendar.HOUR);
    int minute = calendar.get(Calendar.MINUTE);
    int second = calendar.get(Calendar.SECOND);
    if(calendar.get(Calendar.AM_PM) == 0)
      amORpm = "AM";
    else
      amORpm = "PM";
    return "Current Time : " + hour + ":" + minute + ":" + second + " "
+ amORpm;
    }
}

import java.net.MalformedURLException;
import java.rmi.Naming;
import java.rmi.NotBoundException;
import java.rmi.RemoteException;
import java.util.logging.Level;
import java.util.logging.Logger;

public class TimeClient {
    public static void main(String[] args) {
        try {
            String url="//localhost/time";
            TimeOperations look_up =(TimeOperations)
Naming.lookup(url);
            System.out.println(""+look_up.showTime());
        } catch (NotBoundException ex) {
            ex.printStackTrace();
        } catch (MalformedURLException ex) {
            ex.printStackTrace();
        } catch (RemoteException ex) {
            ex.printStackTrace();
        }
    }
}
```