JUST DO IT
HÖHERE ABTEILUNG FÜR
INFORMATIONSTECHNOLOGIE

tgm
Die Schule der Technik

Laborprotokoll

Remote Method Invocation

Systemtechnik Labor
4BHIT 2015/16, Gruppe B

Philip Vonderlind

Version 0.1
Begonnen am 6. Mai 2016
Beendet am 13. Mai 2016

Note:
Betreuer: Prof. Borko

Inhaltsverzeichnis

1 Einführung

Verteilte Objekte haben bestimmte Grunderfordernisse, die mittels implementierten Middlewares leicht verwendet werden können. Das Verständnis hinter diesen Mechanismen ist aber notwendig, um funktionale Anforderungen entsprechend sicher und stabil implementieren zu können.

1.1 Ziele

Diese Übung gibt eine einfache Einführung in die Verwendung von verteilten Objekten mittels Java RMI. Es wird speziell Augenmerk auf die Referenzverwaltung sowie Serialisierung von Objekten gelegt. Es soll dabei eine einfache verteilte Applikation in Java implementiert werden.

1.2 Voraussetzungen

- Grundlagen Java und Software-Tests
- Grundlagen zu verteilten Systemen und Netzwerkverbindungen
- Grundlegendes Verständnis von nebenläufigen Prozessen

1.3 Aufgabenstellung

Folgen Sie dem offiziellen Java-RMI Tutorial, um eine einfache Implementierung des PI-Calculators zu realisieren. Beachten Sie dabei die notwendigen Schritte der Sicherheitseinstellungen (Security-Manager) sowie die Verwendung des RemoteInterfaces und der RemoteException.

Implementieren Sie ein Command-Pattern [2] mittels RMI und übertragen Sie die Aufgaben/Berechnungen an den Server. Sie können am Client entscheiden, welche Aufgaben der Server übernehmen soll. Die Erweiterung dieser Aufgabe wäre ein Callback-Interface auf der Client-Seite, die nach Beendigung der Aufgabe eine entsprechende Rückmeldung an den Client zurück senden soll. Somit hat der Client auch ein RemoteObject, welches aber nicht in der Registry eingetragen wird sondern beim Aufruf mittels Referenz an den Server übergeben wird.

2 Ergebnisse

2.1 Testen des Tutorialcodes

2.1.1 Buildautomation mit Ant

Ant ist ein Bildautomationtool welches auf einer Datei namens build.xml basiert. Hier werden viele Properties, Targets die das Programm umschreiben definiert. Die Properties sind insofern wichtig, da sie Informationen über den Classpath, Java Version, Binaries usw., enthalten. Die angegebenen Targets können dann später nach dem builden mittels der Commandline wie folgt ausgeführt werden.

```
1   ant [target]
```

Listing 1: Ant Beispiel

2.1.2 Testen des simplen Tutorialcodes

Bevor die Sockets die in dem Tutorial verwendet werden funktionieren, muss man noch in der java.policy Datei ein paar Zeilen am Anfang wie folgt hinzufügen, da man sonst eine AccessControlException bei der Ausführung des engine Targets bekommt.

```
    sudo vim /usr/lib/jvm/java-8-openjdk/jre/lib/security/java.policy

    // Inserted to grant all privileges
4   grant codeBase "file:/home/sons/-" {
        permission java.security.AllPermission;
    };
```

Listing 2: Bearbeiten von java.policy

Um ein Projekt mit ant zu builden, geht man einfach per Commandline in den Projektordner mit dem Testcode. Nun kann man mit den ant Targets den Server und Client starten (natürlich in seperaten Terminals). Engine entspricht hierbei dem Server, compute dem Client.

```
    cd ~/School/SYT/Syt_RMI/rmiTutorial
    ant
    ant engine
4   ant compute
```

Listing 3: Builden mit ant

In Eclipse einbinden und dann in der Console mit ant im Projektordner das Projekt builden. Wirft einen Fehler access denied Socket Permission Lösung: im java.policy file die permissions anpassen. Nun ant engine ausführen um den Server zu starten und ant compute um sich pi zu berechnen.

2.1.3 Testen des Command Patterns

Um das vorbereitete Command Patter auszuprobieren, geht man wiederum mc cd in den Ornder mit dem jeweiligen Testcode und builded diesen mit dem ant Befehl.

```
1   cd ~/School/SYT/Syt_RMI/rmiCommandPattern
    ant
```

Listing 4: Builden des Command Pattern Testcodes

Mittels der ant - Befehle "clientünd ßerver", kann man nun die jeweils dem Namen entsprechenden Komponenten starten (in seperaten Terminalinstanzen versteht sich) .

```
ant server
ant client
```

Listing 5: Ausführen des Testcodes des Command Patterns

2.2 Implementierung eines eigenen Command Patterns

Aufgabe war es nun, ein eigenes Command Pattern zu erzeugen, welches am Server die Eulersche Zahl mit dem entsprechenden Algorithmus berechnet und das Ergebnis anschließend ausgibt.

2.2.1 Implemetierung des Command Patterns

Ein Command Pattern besteht aus zwei Interfaces, dem CommandExecutor und dem eigentlichen Command. Das Command Interface schreibt hier vor, dass es eine Methode execute geben muss, welche dann logischerweiße vom CommandExecutor ausgeführt wird.

```
    public interface Command extends Serializable {
        public void execute();
3   }
```

Listing 6: Command Interface

In diesem Beispiel heiSSt der CommandExecutor doSomethindService, ist aber gleich implementiert wie das standartmäßige Interface.

```
    public interface DoSomethingService extends Remote {
2
    public void doSomething(Command c) throws RemoteException;

}
```

Listing 7: Command Executor

Dieses Interface wird nun in der Klasse ServerService konkret verwendet, um das Command auszuführen.

```
    public class ServerService implements DoSomethingService {

    @Override
    public void doSomething(Command c) throws RemoteException {
5       c.execute();

    }

}
```

Listing 8: ServerService Klasse

2.2.2 Implementierung eines Callbacks

Damit der Server dem Client eine Rückmeldung nach der Berechnung schicken kann, hier etwa mit dem Ergebnis der Berechnung, muss man einen sogenannten Callback implementieren. Die Implemetierung passiert über ein Interface mit dem Namen Callback, welches 3 Methoden vorschreibt, nämlich :

- **set** Setzt jenen Wert, welcher übetragen werden soll.
- **print** Schreibt den Wert in die Console.
- **receive** Gibt den Wert des Callbacks zurück.

```
1   public interface Callback<T> extends Remote, Serializable {
    public void set(T argument) throws RemoteException;
    public void print() throws RemoteException;
    public T receive() throws RemoteException;
}
```

Listing 9: Interface für das Callback

Nun habe ich dieses Interface auf eine Klasse CallResult angewandt, welche mit dem Typ Double arbeitet. Diese Klasse hat eine globale Variable result, welche von den 3 vorher erwähnten Methoden verwendet wird.

```
    public class CallResult implements Callback<Double>, Serializable{

    private static Double result;
4
    /**
     * Set the value to be used by the other methods
     */
    @Override
9   public void set(Double result) throws RemoteException {
        // TODO Auto-generated method stub
        this.result = result;
    }

14  /**
     * Prints the result to the console
     */
    @Override
    public void print() throws RemoteException {
19      // TODO Auto-generated method stub
        System.out.println("The result is: "+result);
    }

    /**
24   * Receive the result
     */
    @Override
    public Double receive() throws RemoteException {
        // TODO Auto-generated method stub
29      return null;
    }

}
```

Listing 10: CallResult Klasse

Um den Callback verwenden zu können, müssen noch zwei Zeilen Code in die Client Klasse hinzugefügt werden, welche das Callback Object exportieren. Hierbei wird aber nicht direkt in die Registry von RMI gespeichert, da sonst auch andere Clients auf den Callback zugreifen könnten, es soll aber nur dem Server möglich sein.

```
            // Create a new callback
            Callback callback = new CallResult();

            // Export the callback
            Callback callbackStub = (Callback) UnicastRemoteObject.exportObject(callback,0);
```

Listing 11: Implementierung des Callback am Client

2.2.3 Implementierung des Euler-Algorithmus

In EulerCalc, welches das Interface Calculation implementiert.

```
    public interface Calculation {

    public void calculate();
    public double getResult();
}
```

Listing 12: Calculation Interface

In calculate wird in der konkreten Klasse EulerCalc eine weitere Methode calcEuler aufgerufen, welche den Algorithmus zu Berechnung der eulerschen Zahl beinhaltet (diesers orientiet sich and er Summenformel welche die eulersche Zahl definiert). Diese sieht wie folgt aus:

```
    private void calcEuler(int digits){

        // Buffer for the current e
        double buffer = 1.0;

        // current value of the fraction in the limes
        double currentNumber;

        // The factorial of the current fraction in the limes
        double factorial = 1.0;

        // This loop runs until the amount of digits the user
        // whiched for is reached
        for(int i=1; i < digits;i++){

            // Multiply the factorial with the counter, so it
            // behaves like a mathematical factorial
            factorial = factorial * i;

            // By dividing 1 through the current factorial you
            // get the digit at count i for the number e
            currentNumber = 1 / factorial;

            // add the current number to the buffer
            buffer += currentNumber;
        }

        // Set global variable result to the value of buffer
        result = buffer;
    }
```

Listing 13: Algorithms der eulerschen Zahl

Mit getResult kann man schließlich auf das Ergebnis dieser Berechnung zurückgreifen.

2.2.4 Implementieren des CalculationCommands

Hier passiert eigentlich die ganze "Magie"des Callbacks und des Commands. Mittels zwei Parametern werden sowohl ein Callback, als auch ein Calculation Objekt übergeben und auf die Attribute der Klasse geschrieben. In der execute Methode wird nun das Ergebnis berechnet, und dieses dann mit set auf den Callback geschrieben und dann mit print an den Client gesendet.

Wieder in der Client Klasse man noch das Command aufrufen, mit dem voher gesetzen Callback und einem neuen EulerCalc Objekt als Parameter.

```
Command calcEuler = new CalculationCommand(new EulerCalc(), callback);
```

Listing 14: Aufrufen des Commandos

2.3 Quellen

1. "The Java Tutorials - Trail RMI"; online: `http://docs.oracle.com/javase/tutorial/rmi/`
2. "Command Pattern"; Vince Huston; online: `http://vincehuston.org/dp/command.html`
3. "Beispiel Konstrukt für Command Pattern mit Java RMI"; Michael Borko; online: `https://github.com/mborko/code-examples/tree/master/java/rmiCommandPattern`
4. GitHub der Aufgabe; online: `https://github.com/pvonderlin-tgm/Syt_RMI`

2.4 Zeitaufzeichnung

Datum	Dauer	Beschreibung
6.5.2016	4 Stunden	RMI - Tutorials ausführen
12.5.2016	2 Stunden	Callback und Protokoll fertigstellen

Listings

