Client-/Server-Anwendung “Wetterdienst” mit JAVA-RMI

Sebastian Schmid S0543196

Übung 3 Verteilte Systeme

Aufgabenstellung

Zu erstellen waren ein Client-Server Protokoll mit zusätzlichen Callbackfunktionen mit Implementierung in JAVA-RMI.

Dabei sollte vom Client ein Datum an den Server gesendet werden. Der Server sollte Temperaturwerte zum gegebenen Datum in einer CSV-Datei nachschlagen, min, max und avg berechnen und an den Client zurücksenden. Für etwaige Änderungen sollte es dem Client möglich sein via Callbacks Änderungen der Daten anzufragen.

Das Emfangene solle dann vom Client im Terminal angezeigt werden.

Lösung

RMI_ServerInterface

```
1  package org.geblubber.sebsch.network.server;
2
3  import org.geblubber.sebsch.network.client.RMI_ClientCallbackInterface;
4
5  import java.rmi.Remote;
6  import java.rmi.RemoteException;
7
8  public interface RMI_ServerInterface extends Remote {
9
10     String getTemperatureList(String day) throws RemoteException;
11
12     void registerForCallback(RMI_ClientCallbackInterface callbackClientObject) throws
           java.rmi.RemoteException;
13
14     void unregisterForCallback(RMI_ClientCallbackInterface callbackClientObject) throws
           java.rmi.RemoteException;
15 }
```

Für den Client wird die Methode getTemperatureList(String day) bereitgestellt. Außerdem werden für die Registrierung der Callbacks noch zwei Methoden benötigt.

Der Client greift über Reflection auf die Methoden dieses Interface als Stub zu.

RMI_Server

```java
import org.geblubber.sebsch.network.client.RMI_ClientCallbackInterface;

import java.rmi.RemoteException;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import java.rmi.server.UnicastRemoteObject;

public class RMI_Server implements RMI_ServerInterface {


    (...)

     public RMI_Server(int port) {
        this.csvFILE = ("temps.csv");

        try {
            RMI_ServerInterface stub = (RMI_ServerInterface)
                UnicastRemoteObject.exportObject(this, 0);

            LocateRegistry.createRegistry(port);
            Registry registry = LocateRegistry.getRegistry();
            registry.bind("RMI_ServerInterface", stub);

            System.err.println("Server ready");

        } catch (Exception e) {
            System.err.println("Server exception: " + e.toString());
            e.printStackTrace();
        }
    }

    (...)

    @Override
    public void registerForCallback(RMI_ClientCallbackInterface callbackClientObject) throws
        RemoteException {
        this.client = callbackClientObject;
        System.out.println("Client für Callback registriert.");
        doCallback();
    }

    private void doCallback() throws RemoteException {
        csvParser csvdate2 = new csvParser(this.csvFILE, this.query);
        System.out.println("Callback wird ausgeführt");

        RMI_ClientCallbackInterface cli = (RMI_ClientCallbackInterface) this.client;
        cli.notifyMe(this.dayTab1.compare(this.dayTab1));
    }

    @Override
    public void unregisterForCallback(RMI_ClientCallbackInterface callbackClientObject)
        throws RemoteException {
        this.client = null;
        System.out.println("Client für Callback entfernt.");
```

```
    }

   (...)

}
```

Im Konstruktor wird die Registry erstellt und darüber das stub bereitgestellt.

RMI_Client

```
package org.geblubber.sebsch.network.client;

import org.geblubber.sebsch.network.server.RMI_ServerInterface;

import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import java.util.Scanner;

public class RMI_Client {

    public RMI_Client() {
        new RMI_Client(null);
    }

    public RMI_Client(String host) {

        try {

            Registry registry = LocateRegistry.getRegistry(host);
            RMI_ServerInterface stub = (RMI_ServerInterface)
                registry.lookup("RMI_ServerInterface");

            String response = stub.getTemperatureList(uii());
            System.out.println(response);

            if (!response.equals("Call is not valid! Please send the date in this format:
                YYYY-MM-DD\n")) {
                RMI_ClientCallbackInterface callBackObj = new RMI_ClientCallback();
                stub.registerForCallback(callBackObj);
                Thread.sleep(3000);
                stub.unregisterForCallback(callBackObj);
            }

        } catch (Exception e) {
            System.err.println("Client exception: " + e.toString());
            e.printStackTrace();
        }
    }

   (...)
}
```

Der Client fragt über ein Lookup bei localhost nach der gewünschten Registry an. Über diese wird dann die freigegebene Methode des Servers aufgerufen und mit den Parametern versehen.

Direkt danach wird der Callback ausgeführt. Hierbei drehen sich die “Rollen” Server / Client scheinbar um.

RMI_ClientCallbackInterface

```
1  package org.geblubber.sebsch.network.client;
2
3  import java.rmi.RemoteException;
4  import java.rmi.server.UnicastRemoteObject;
5
6
7  public interface RMI_ClientCallbackInterface extends java.rmi.Remote {
8      public String notifyMe(String message) throws java.rmi.RemoteException;
9  }
```

Der Client stellt jetzt ein Stub zur Verfügung. Dieses wird vom Client an den Server als Parameterübergabe von registerForCallback() übergeben.

RMI_ClientCallback

```
1  package org.geblubber.sebsch.network.client;
2
3  import java.rmi.RemoteException;
4  import java.rmi.server.UnicastRemoteObject;
5
6
7  public class RMI_ClientCallback extends UnicastRemoteObject implements
       RMI_ClientCallbackInterface {
8
9      public RMI_ClientCallback() throws RemoteException {
10         super();
11     }
12
13     public String notifyMe(String message) {
14         String returnMessage = "Call back received: \n" + message;
15         System.out.println(returnMessage);
16         return returnMessage;
17     }
18
19 }
```

Die Methode notifyMe wird vom Server aufgerufen. Dieser sendet seine bei Bedarf berechneten Werte und gibt sie damit auf Clientseite aus.