Verteilte Systeme und Kommunikationsnetze

Testumgebung

Praktikum 4

Fachhochschule Bielefeld
Campus Minden
Studiengang Informatik

Beteiligte Personen:

Name	Matrikelnummer
Christian Krebel	1151165
Dennis Petana	1157886
Hannes Rüffer	1151954

Aufgaben:

Aufgabe	Gelöst
Chat-System mit Java RMI	komplett

---

21. November 2018

Inhaltsverzeichnis

Bearbeitung der Aufgaben

1 Aufgabe - Chat-System mit Java RMI

Aufgabenstellung

Sie sollen ein Chat-System unter Verwendung von Java RMI implementieren. Das System besteht aus einem Server, an den beliebig viele Clients angeschlossen werden können. Das System soll nach bekanntem Prinzip arbeiten: Nachrichten werden auf einem Client eingegeben und über den Server an alle angeschlossenen Clients weitergeleitet. Es gibt zwei prinzipielle Möglichkeiten, ein solches System zu realisieren:

a) Die Nachrichten werden vom Server vorgehalten. Verbundene Clients überprüfen in zyklischen Abständen das Vorliegen neuer Nachrichten und rufen diese ab (Polling).

b) Clients registrieren im Server jeweils ein sog. Callback-Objekt. Dieses beinhaltet Methoden zum Empfang und zur Anzeige von Nachrichten.

Verwenden Sie für Ihre Lösung den letztgenannten Ansatz und **protokollieren Sie Ihr Vorgehen mit hilfe der Vorlage**. Es sind hierzu zwei Klassen von Remote-Objekten vorzusehen:

- Die Klasse `ChatServer` stellt nach außen eine Methode zur Verfügung, mit der sich Clients beim Server unter einer Referenz auf das Client-Callback-Objekt (ClientProxy) registrieren und de-registrieren können:

  ```
  public interface ChatServer extends Remote {
   public ChatProxy subscribeUser (ClientProxy handle)
    throws RemoteException;
   public boolean unsubscribeUser (ClientProxy handle)
    throws RemoteException;
  }
  ```

  Außerdem verteilt der Server die Nachrichten an die Clients. Überlegen Sie sich dazu eine passende Methode!

- `ClientProxy` repräsentiert den Client im Server und unterstützt eine Methode zum Empfang von Nachrichten:

  ```
  public interface ClientProxy extends Remote {
   public void receiveMessage (String username,
    String message) throws RemoteException;
  }
  ```

Die entsprechenden Implementierungsklassen realisieren die Schnittstellen-Methoden sowie weitere Methoden. Die Implementierung der Schnittstelle `ChatServer` erzeugt in der main() Routine ein neues Objekt von sich selbst und registriert das Objekt dann in der **RMI Registry** unter einem frei wählbaren Namen. Daneben benötigen Sie noch einen Client (GUI oder interaktive Shell). Dieser liest eine Referenz auf das `ChatServer`-Objekt aus der **Registry** aus und ermöglicht es einem User, sich unter Angabe eines Namens per `subscribeUser()` Methode beim Chat zu registrieren.
Testen Sie Ihr System, in dem Sie einen Chat mit mehreren Clients aufsetzen.

Vorbereitung

Zuallererst wurde gebrainstormed was der `ChatProxy` machen soll. Dazu wurden Stichpunkte aufgeschrieben, mit denen man dann gezielter nach Dokumentationen/Hilfe im Internet finden konnte. Außerdem haben wir nach Dokumentationen zu RMI gesucht und gefunden[1][2]. Wir haben uns entschieden die Eingabe und Ausgabe auf der Konsole stattfinden zu lassen.

Durchführung

Nachdem wir alle nötigen Informationen recherchiert haben, wurden das Projekt und die Klassenrümpfe erstellt. Begonnen wurde mit der Klasse `SimpleChatServer`. Der überwiegende Part konnte mithilfe der Vorlesung implementiert werden.
Man sollte darauf aufpassen, dass alle Parameter und Rückgabewerte selbst auch Remote-Objekte sein müssen[3].
Die Methode `broadcastMessage` schickt allen Benutzer einen Username und eine Nachricht.
Die Klasse `SimpleChatProxy` erzeugt eine Instanz, die einem jeden Benutzer gegeben wird, der sich am Server anmeldet. Über diese Instanz können Nachrichten an Server bzw. andere Benutzer verschickt werden.

```
1 // In SimpleChatServer
  ChatProxy someProx = new SimpleChatProxy(this);
3 ...
  // In SimpleClientProxy
5 stub1.setChatProxy(server.subscribeUser(stub1));
```

1 https://docs.oracle.com/javase/7/docs/technotes/guides/rmi/index.html
2 https://www.tutorialspoint.com/java_rmi/java_rmi_introduction.htm
3 https://stackoverflow.com/questions/11049810/rmi-notserializableexception-although-its-a-remote-object

In der Klasse `SimpleClientProxy` wurden die nötigen Methoden so implementiert, wie es logisch ist: `recieveMessage` gibt die Nachricht auf der Konsole aus, `sendMessage` schickt eine Nachricht über den `ChatProxy` an den Server und `setChatProxy` registriert den `ChatProxy` für einen Client. Nach dem Erstellen eines `SimpleClientProxy` wird dieser zu einem `UnicastRemoteObject` exportiert. Über die Registry und dem gleichen Name, den man auch im `SimpleChatServer` verwendet hat, bekommt man den Server, auf dem sich der Client dann registrieren kann.

Fazit

Probleme hatten wir zuerst damit, dass wir nicht genau wussten was der ChatProxy ist bzw. was diese Klasse tun soll. Dies konnte aber durch Brainstorming erledigt werden: Sie ist lediglich dafür da, dass der Client eine Nachricht an den Server schicken kann. Alle Parameter und Rückgabewerte müssen auch selbst Remote-Objekte sein, ansonsten gibt es einen `RuntimeError` bei einem remote-Methodenaufruf. Alle Methoden der Interfaces, die Remote implementieren, müssen RemoteException werfen, denn sonst wird eine Implementationsklasse eines Remote-Interfaces nicht als valides Remote-Objekt betrachtet. In dem Fall würde es zu einem `RuntimeError` beim Aufruf der Methode `UnicastRemoteObject.exportObject()` führen.