Entwicklung verteilter Anwendungen in Java

Wintersemester 2022/2023

Jan Zimmer

Im Folgenden soll der Aufbau und die Funktionsweise eines einfachen, konsolenbasierten TCP Chat Anwendung, die im Rahmen der Veranstaltung erstellt wurde, dokumentiert werden.

Bei dem Programm handelt es sich um eine Client-Server Anwendung, geschrieben in Java. Sie setzt sich im wesentlichen aus zwei äußeren Klassen, dem ***Client*** und dem ***Server***. Beide verfügen respektive über die inneren Klassen ***InputHandler*** und ***ConnectionHandler***.

Der ***Server*** verwendet einen Thread-Pool, um eingehende Verbindungen von Clients zu bearbeiten und die Kommunikation mit ihnen zu verwalten. Er verwaltet auch die Benutzernamen, die in einer lokalen Datei gespeichert werden und das Versenden von Nachrichten an alle verbundenen Clients (Broadcasting).

Wenn die Serverklasse ausgeführt wird, erstellt sie eine neue Instanz eines ***ExecutorService-Threadpools*** und einen ***ServerSocket*** am angegebenen Port. Die Server-Klasse tritt dann in eine Schleife ein, die auf eingehende Verbindungen an dem angegebenen Port wartet. Jede eingehende Verbindung wird von einer neuen Instanz der ***ConnectionHandler***-Klasse behandelt, die zu einer ***ArrayList*** von Verbindungen hinzugefügt wird. Die ***ConnectionHandler***-Instanzen werden dann vom Threadpool ausgeführt.

Die Klasse Server lädt Benutzernameninformationen (Nicknames) aus einer ***Hashmap***, die in einer Datei namens "***userList.lst***" gespeichert wird. Die Klasse außerdem über Methoden, um Nachrichten an alle verbundenen Clients zu senden und den Server im Fehlerfall kontrolliert zu beenden.

Methoden:

main(String[] args): der Einstiegspunkt des Programms. Sie erzeugt eine neue Instanz der Klasse Server und ruft die Methode ***run()*** auf. Wenn ein Befehlszeilenargument angegeben wird, wird dieses als Portnummer verwendet, andernfalls wird der Standardport 10101 verwendet.

run(): startet den Server. Sie initialisiert den Thread-Pool und den Welcome-Socket, wartet auf eingehende Verbindungen und verarbeitet diese, indem sie neue Instanzen der ***ConnectionHandler***-Klasse erstellt und diese im Threadpool ausführt.

shutDownServer(): stoppt die Hauptschleife innerhalb der ***run()***-Methode, schließt den Server und beendet jeden Verbindungshandler in der Verbindungsliste.

broadcast(String s): sendet den angegebenen String an alle verbundenen Clients.

loadUserMap(): lädt die Benutzerdaten aus der Datei "userList.lst" in die userMap Hashmap.

checkForUserMapFile(): erstellt die Datei "userList.lst", wenn sie nicht vorhanden ist.

Felder:

threadpool: eine Instanz der ExecutorService-Klasse, die zur Verwaltung des Threadpools verwendet wird.

port: eine int-Variable, die den Port angibt, an dem der Server lauschen wird.

server: eine Instanz der Klasse ServerSocket, die den Server-Socket darstellt.

connections: eine ArrayList von ConnectionHandler-Objekten, die die verbundenen Clients darstellen.

finished: eine boolesche Variable, die zum Anhalten der Hauptschleife in der run()-Methode verwendet wird.

userFile: ein File-Objekt, das die Datei "userList.lst" darstellt.

userMap: eine HashMap, die die Namen der Benutzer enthält.

userMapLock: ein ReentrantLock-Objekt, das zur Synchronisierung des Zugriffs auf das userMap-Feld verwendet wird.

Der ***ConnectionHandler*** verwaltet die Kommunikation mit einem einzelnen, mit dem Server verbundenen Client. Sie wird von der Klasse Server verwendet, um mehrere Verbindungen von verschiedenen Clients zu verwalten. Diese Klasse ist für die Verarbeitung der vom Client empfangenen Nachrichten und das Senden von Nachrichten an den Client zuständig.

Wenn eine Instanz der Klasse ConnectionHandler erstellt wird, wird ihr ein ***Socket***-Objekt übergeben, das die Verbindung zum Client darstellt. Die Klasse instanziiert dann ein ***PrintWriter***-Objekt, das zum Senden von Nachrichten an den Client verwendet wird, und ein ***BufferedReader***-Objekt, das zum Empfangen von Nachrichten vom Client verwendet wird.

Die Klasse ConnectionHandler tritt dann in eine Schleife ein, die auf eingehende Nachrichten vom Client wartet und auf der Grundlage der empfangenen Nachricht entsprechende Maßnahmen ergreift. Die Klasse verfügt auch über eine Methode zum Beenden der Verbindung und Schließen des Sockets.

Methoden:

run(): Startet den ConnectionHandler. Er tritt in eine Schleife ein, die auf eingehende Nachrichten vom Client wartet und auf der Grundlage der empfangenen Nachricht entsprechende Aktionen durchführt.

shutDownConnectionHandler(): schließt den Socket und beendet den ConnectionHandler.

handleCommand(String userInput): Interpretiert ein vom Client empfangenes Kommando (:command) und ruft die entsprechende Methode auf.

changeNickname(String userInput): Ändert den Nickname eines Clients auf den angegebenen Nickname.

isKnownUser(): Gibt Wahr zurück, wenn der verbundene Client bereits mit einem Nicknamen aus der Nutzerliste assoziiert ist, und Falsch, wenn dies nicht der Fall ist.

askForNickname(): Fragt einen unbekannten Nutzer nach seinem gewünschten Nickname und fügt ihn der Liste hinzu.

messageToClient(String s): Sendet eine Nachricht an den Client, die auf seiner lokalen Konsole ausgegeben wird.

printConnectedUsers(): Sendet eine Nachricht mit allen derzeit verbundenen Nutzern an den Client.

printHelp(): Sendet eine Nachricht mit den verfügbaren Kommandos (beginnend mit einem Doppelpunkt) an den Client.

Felder:

client: ein Socket-Objekt, das die Verbindung zum Client darstellt.

outputToClient: ein PrintWriter-Objekt, das zum Senden von Nachrichten an den Client verwendet wird.

inputFromClient: ein BufferedReader-Objekt, das verwendet wird, um Nachrichten vom Client zu empfangen.

isDone: eine boolesche Variable, die angibt, ob der Verbindungshandler die Ausführung beenden soll.

userMapLock: ein ReentrantLock-Objekt, das zur Synchronisierung des Zugriffs auf die Benutzerkarte verwendet wird.

Die Klasse ***Client*** ist das lokal auszuführende Gegenstück zum Server. Sie stellt eine Verbindung zu einem Server mit einer bestimmten IP-Adresse und einem bestimmten Port her und ermöglicht die Kommunikation mit dem Server. Sie enthält auch einen Mechanismus zur Wiederherstellung der Verbindung zum Server, wenn die Verbindung unerwartet geschlossen wird.

Wenn die Client-Klasse ausgeführt wird, erstellt sie ein neues ***Socket***-Objekt und stellt eine Verbindung mit dem Server an der angegebenen IP-Adresse und dem Port her. Außerdem instanziiert sie ein ***PrintWriter***-Objekt, das zum Senden von Nachrichten an den Server verwendet wird, und ein ***BufferedReader***-Objekt, das zum Empfangen von Nachrichten vom Server verwendet wird.

Die Client-Klasse instanziiert einen ***InputHandler*** in einem eigenen Thread, der die Eingaben des Benutzers verarbeitet und an den Server sendet. Die Client-Klasse tritt dann in eine Schleife ein, die auf eingehende Nachrichten vom Server wartet und diese auf der Konsole ausgibt. Wenn die Verbindung zum Server unerwartet beendet wird, versucht der Client bis zu fünfmal, die Verbindung zum Server wiederherzustellen, bevor er sich beendet.

Methoden:

main(String[] args): der Einstiegspunkt des Programms. Sie erstellt eine neue Instanz der Client-Klasse und ruft die Methode ***run()*** auf. Wenn Befehlszeilenargumente angegeben wurden, wird das erste als IP-Adresse und das zweite als Portnummer verwendet, andernfalls werden die Standard-IP-Adresse "10.0.3.103" und Port 10101 verwendet.

run(): startet den Client und stellt eine Verbindung zum Server durch den Aufruf der connect() Methode her.

connect(): erstellt ein neues Socket-Objekt und stellt eine Verbindung zum angegebenen Server her und instanziiert die Objekte PrintWriter und BufferedReader. Ein neuer Thread für die InputHandler-Klasse wird ebenfalls gestartet.

reconnect(): versucht, die Verbindung zum Server wiederherzustellen, wenn die Verbindung unerwartet geschlossen wird.

shutdownClient(): beendet die Ausführung des Client-Programms.

Felder:

port: eine int-Variable, die den Port angibt, an dem der Server lauscht.

ipadr: eine String-Variable, die die IP-Adresse des Servers angibt.

client: ein Socket-Objekt, das die Verbindung zum Server darstellt.

outputToServer: ein PrintWriter-Objekt, das zum Senden von Nachrichten an den Server verwendet wird.

inputFromServer: ein BufferedReader-Objekt, das verwendet wird, um Nachrichten vom Server zu empfangen.

isDone: eine boolesche Variable, die angibt, ob der Client die Ausführung beenden soll.

isReconnectAttempt: eine boolesche Variable, die angibt, ob der Client gerade versucht, sich erneut mit dem Server zu verbinden.

inputHandlerThread: ein Thread-Objekt, das den Thread für die InputHandler-Klasse darstellt.

Der ***InputHandler*** ist für die Verarbeitung der Eingaben des Benutzers und deren Weiterleitung an der Server zuständig. Dazu liest sie Eingaben von der Konsole und sendet sie an den Server-Socket.

Wenn eine Instanz der InputHandler-Klasse erstellt wird, instanziiert sie ein ***BufferedReader***-Objekt, das zum Lesen von Eingaben von der Konsole verwendet wird.

Die InputHandler-Klasse tritt dann in eine Schleife ein, die auf eingehende Eingaben des Benutzers wartet und diese an den Server sendet.

Methoden:

run(): Startet den InputHandler. Er tritt in eine Schleife ein, die auf eingehende Eingaben des Benutzers wartet und diese an den Server sendet.